NPS52-85-010

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AN INTEGRATED SYSTEMS ARCHITECTURE FOR REAL-TIME CONTOUR SURFACE DISPLAY GENERATION

Robert A. Walker

Michael J. Zyda

August 1985

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R.H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

This report was prepared by:

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1 REPORT NUMBER: NPS52-85-010

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4 TITLE (and Subtitle): AN INTEGRATED SYSTEMS ARCHITECTURE FOR REAL-TIME CONTOUR SURFACE DISPLAY GENERATION

5. TYPE OF REPORT & PERIOD COVERED

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): Robert A. Walker
Michael J. Zyda

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School
Monterey, CA 93943-5100

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61152N; RR000-01-NP
N0001485WR41005

11 CONTROLLING OFFICE NAME AND ADDRESS: Chief of Naval Research
Arlington, VA 22217

12. REPORT DATE: August 1985

13. NUMBER OF PAGES: 59

14 MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16 DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

General Terms: Algorithms, architecture
Key Words and Phrases: contouring, contouring tree, contour surface display generation, real-time display generation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We present in this study an integrated systems design for a VLSI multiprocessor capable of generating contour surface displays in real-time. We begin by examining a proposed architecture of one such contour surface display generator, and the particular application that initiated the design of that architecture (1). In the course of the discussion on the original architecture, we detail problems discovered with that architecture. We then propose a new architecture for the contour surface display generator that is both precise in detail, and more technologically realistic.

An Integrated Systems Architecture for Real-Time Contour Surface Display Generation ‡

Robert A. Walker and Michael J. Zyda

Naval Postgraduate School,

Code 52, Dept. of Computer Science,

Monterey, California 93943-5100

ABSTRACT

We present in this study an integrated systems design for a VLSI multiprocessor capable of generating contour surface displays in real-time. We begin by examining a proposed architecture of one such contour surface display generator, and the particular application that initiated the design of that architecture [1]. In the course of the discussion on the original architecture, we detail problems discovered with that architecture. We then propose a new architecture for the contour surface display generator that is both precise in detail, and more technologically realistic.

Categories and Subject Descriptors: I.3.1 [**Hardware Architecture**]: architectures, parallel processing, VLSI implementations; I.3.2 [**Graphics Systems**]: multiprocessing systems; I.3.5 [**Computational Geometry and Object Modeling**]: data structures, discrete planar contours, modeling molecules, surface approximation, surface generation, surface representation, surfaces, 3D graphics; I.3.6 [**Methodology and Techniques**]: contouring, interactive systems, parallel processing; I.3.7 [**Three-Dimensional Graphics and Realism**]: line drawings, line generation algorithms, real-time graphics, surface plotting, surface visualization, surfaces; I.3.m [**Miscellaneous**]: VLSI;

General Terms: Algorithms, architecture;

Additional Key Words and Phrases: contouring, contouring tree, contour surface display generation, real-time display generation;

1. Introduction

Contour surface display generation is one of the most frequently used applications graphics algorithms [2-10]. A contour surface display is a visual representation of a surface by the

‡ This work has been supported by the NPS Foundation Research Program.

collection of lines formed when that surface is intersected by a set of parallel planes (Figure 1). The lines formed on each of those planes are called contours (Figure 2). A contour represents the set of points that belong to both the surface and the particular intersecting plane. Contour surface displays are used in X-ray crystallography, computer-aided tomography, and other applications for which grid data is collected. Contour surface display generation is generally depicted as a computationally slow operation whose output is sent to a plotter or film recorder. A recent publication has described an architecture, and produced a feasibility determination for a VLSI based contour surface display generator [1]. The architecture proposed in that study, while appropriate for its level, does not provide enough detail for actual implementation. Neither does that study consider some of the issues that occur when one actually designs a working system. This study attempts to remedy those deficiencies by providing an integrated systems architecture for a real-time contour surface display generator that is both precise in detail, and technologically realistic.

1.1. Contour Surface Display Generation is Slow

Our initial premise for this study is that contour surface display generation on a single processor system, such as a graphics workstation with a floating point accelerator, is too slow to be of any use for interactive applications requiring such a display. This premise is based upon [1], and is reinforced by statements found in the literature. A number of papers have been written documenting "breakthroughs" that increase the speed of contour surface display generation. One author has reported that his contour surface display generation subroutine used one second of central processor time on NCAR's Control Data 7600 [4]. Although a contour surface display generation program of this speed is useful for static situations, it is unacceptable for applications that generate a succession of contour surface displays in response to contour level changes read from a control dial. Such a program requires that a new contour surface display be generated, distributed, and displayed in real-time, typically one-thirtieth of a second for current display technology [11].

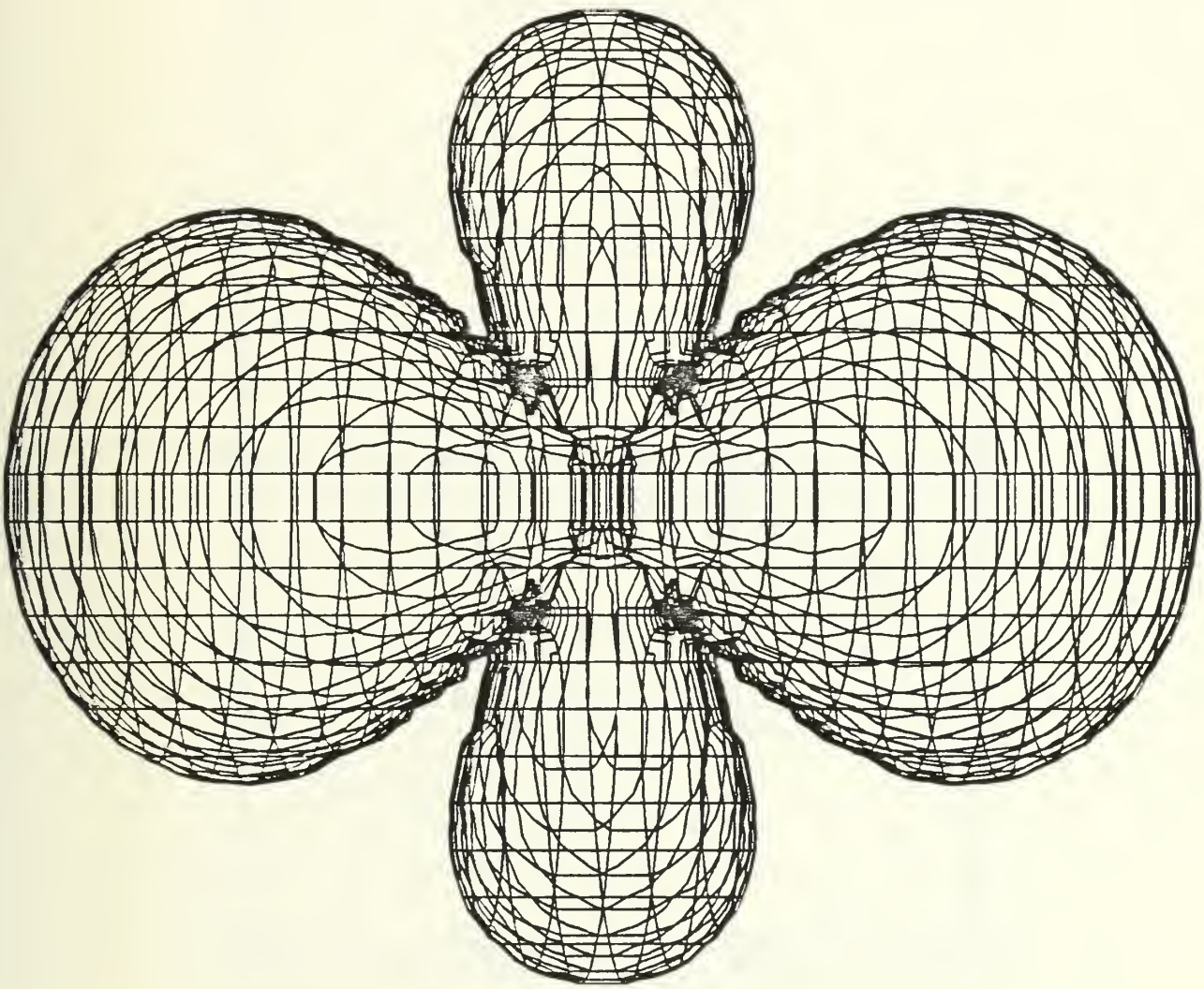

Figure 1
Contour Surface Display Generated from a Hydrogen Atom
Wavefunction Squared (3dz2 orbital)

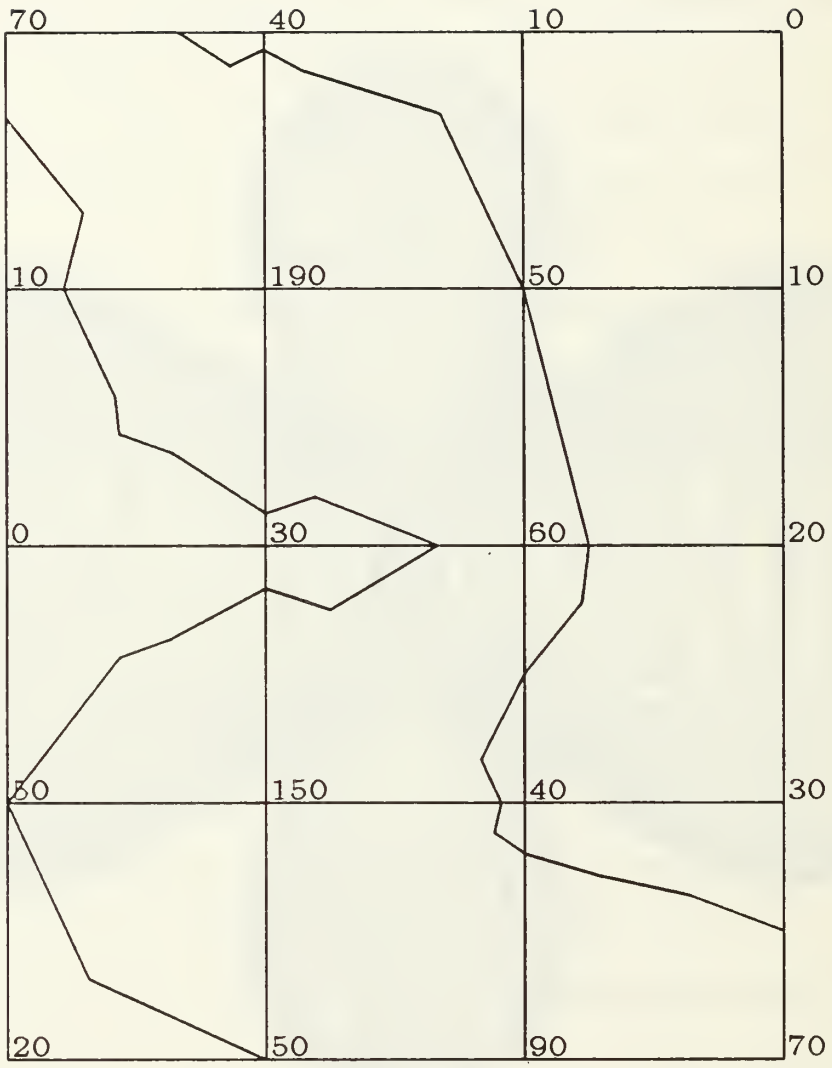

Figure 2a
Example Contour Grid with Contours Drawn for Level 50

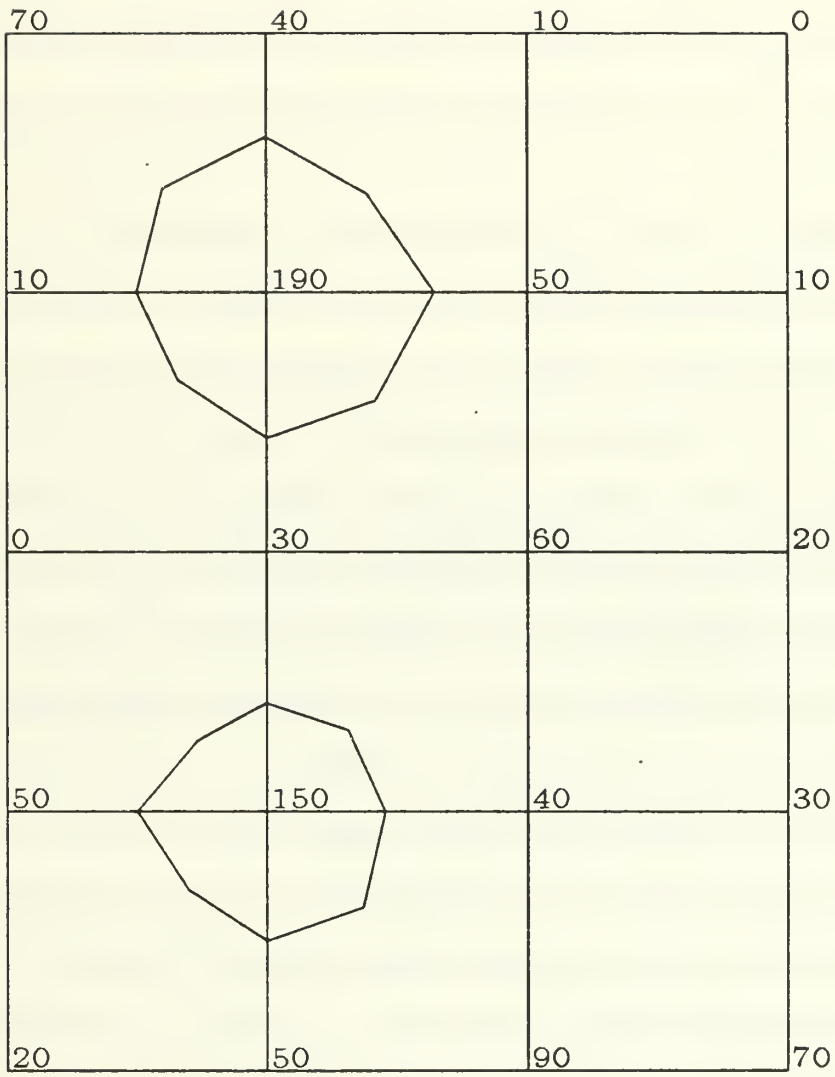

Figure 2b
Example Contour Grid with Contours Drawn for Level 100

One application in which real-time contour surface display generation is important is the determination of molecular structures from the electron density data generated by X-ray crystallography [2]. Such an operation is executed interactively by using a computer graphics program that displays a Dreiding (stick) model of the molecule inside a contour surface display of the corresponding region of the molecule's electron density grid. In addition to the graphics function, the computer program monitors a series of signals generated by the user, while the user is turning the various knobs on a control console [12]. The values read from these knobs are interpreted by the program as modifications to either the molecule or the surface display. Modifications to the molecule take the form of bond rotations or bond lengthenings. Modifications to the contour surface display take the form of an increase or decrease of the contour level. The goal of this process is to produce the stick model of the molecule that best fits inside the given electron density data set. The user can determine whether or not the model fits the density grid by modifying the contour level, shrinking the contour surface to the molecule. Similarly, the user can expand the contour surface from the stick model for better visibility. This function requires that the hardware have the capability to rapidly change the contour display as its contour level changes.

We know from [5] that the generation of a contour surface display, such as those required by the above application, cannot be accomplished in real-time using a conventional uniprocessor. This failure is due to the fact that contour surface display generation algorithms require many more instructions executed per second than can be provided by currently available uniprocessors. In the past, this limitation of the conventional processor has relegated such applications to either the non real-time environment (waiting a few minutes for each display), or to the equally unsatisfying environment of motion picture film. Because of this, the author of [1] looked for a multiprocessor solution to the real-time contour surface display generation problem. At the time of that study, efficient multiprocessor solutions meant VLSI solutions. Consequently, the multiprocessor architecture proposed in that study was qualified by the need for implementation in VLSI. This study continues with that qualification.

1.2. Contouring Definitions

A contour surface is a visual display that represents all points in a particular region of three-space <x,y,z> which satisfy the relation f(<x,y,z>)=k, where k is a constant known as the contour level (Figure 1). The function f represents a physical quantity which is defined over the three-dimensional volume of interest. The visual display created by this algorithm is the collection of lines that belong to the intersection of both the set of points that satisfy the relation f(<x,y,z>)=k, and a set of regularly spaced parallel planes that pass through the region of three-space for which the relation is defined.

For this study, the function f is approximated by a discrete, three-dimensional grid created by sampling that function over the volume of interest. The three-dimensional grid contains a value at each of its defined points that corresponds to the physical quantity obtained from the function, i.e. the value associated with point (x_0,y_0,z_0) is v_0, where $f(x_0,y_0,z_0)=v_0$.

A decomposable algorithm for contour surface display generation is described in [6]. That algorithm is constructed from a two-dimensional contouring algorithm that is used to contour all the possible planar, orthogonal, two-dimensional grids of a larger three-dimensional grid. The two-dimensional contouring algorithm of that study is comprised of components, called algorithm components, that operate on individual 2 x 2 subgrids of a larger two-dimensional grid. (Note: a 2 x 2 subgrid is defined to be that portion of the two-dimensional grid bounded by four adjacent grid points.) In the algorithm, the computations necessary for generating the contour lines for a single 2 x 2 subgrid are independent from those required for any other 2 x 2 subgrid. If we compute the contours corresponding to contour level k for all 2 x 2 subgrids of a two-dimensional grid, then we will have determined the complete set of contours for that grid. If we compute the contours corresponding to contour level k for all possible 2 x 2 subgrids of the larger three-dimensional grid, then we will have the complete contour surface display for that grid. The assemblage of the contours created by this process, i.e. the simultaneous display of all the contours created for all 2 x 2 subgrids of the larger three-dimensional grid, produces a "chicken-wire-like" contour surface display (Figure 1). The full development of this algorithm can be found in

[5-7]. We refer to the results of those studies, and consequently, do not cover the algorithm here in great detail.

2. The Original Architecture

The architecture specified in [1] is described in terms of two basic pieces: the algorithm component processor (Figure 3), and the interconnections necessary between those processors and the graphics system (Figure 4). The algorithm component processor is depicted by way of its expected operations capability, and not by a large amount of actual architectural detail. The interconnections between the algorithm component processors, and the connections to the graphics system are described with somewhat more detail but are still not sufficient for an actual implementation.

2.1. Architecture for the Algorithm Component Processor

The algorithm component processor of [1] is described as having the functionality and complexity of a general purpose microprocessor, the Motorola MC68000. Figure 3 is an overview diagram of that processor. In that figure the important architectural pieces of the processor and their interconnections are depicted. The pieces shown are found in most processors. Instead of giving a detailed specification for those pieces, reference [5] discusses only the important size parameters, and important uses for that hardware when they differ from the norm found in processors of the MC68000 class.

The key parts of Figure 3 are best understood if we describe the operations expected of the algorithm component processor. There are only four: (1) reset the entire system of algorithm component processors, (2) accept a 2 x 2 subgrid description into a particular algorithm component processor, (3) place the coordinates generated for a particular 2 x 2 subgrid onto the system bus, and (4) generate the contours for the 2 x 2 subgrid held in the algorithm component processor. The first operation, the reset operation for the entire system of algorithm component processors, is clearly required. Computing systems are never constructed without some mechanism for providing a known initial state of the hardware.

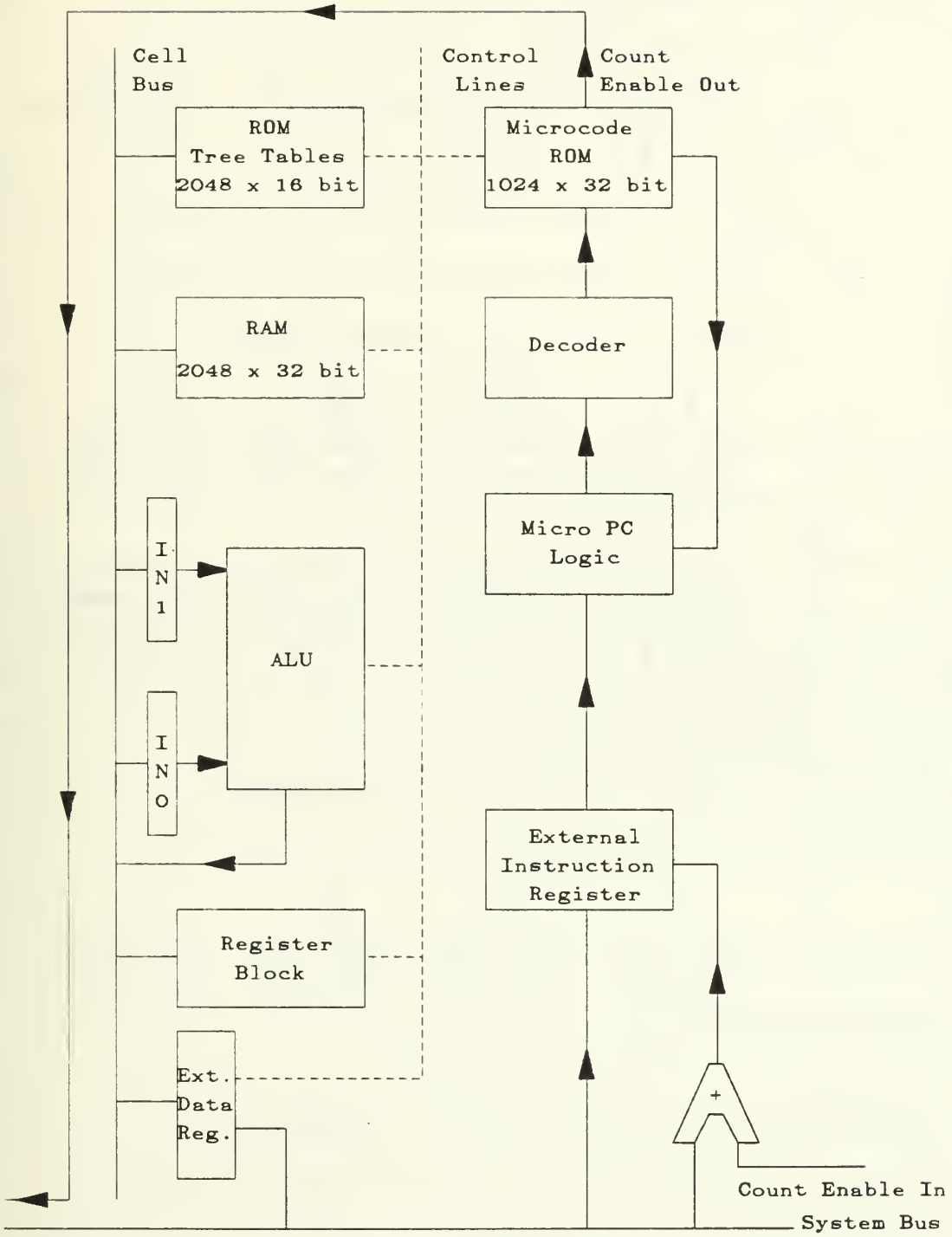

Figure 3.
Block Diagram of the Algorithm Component Processor.

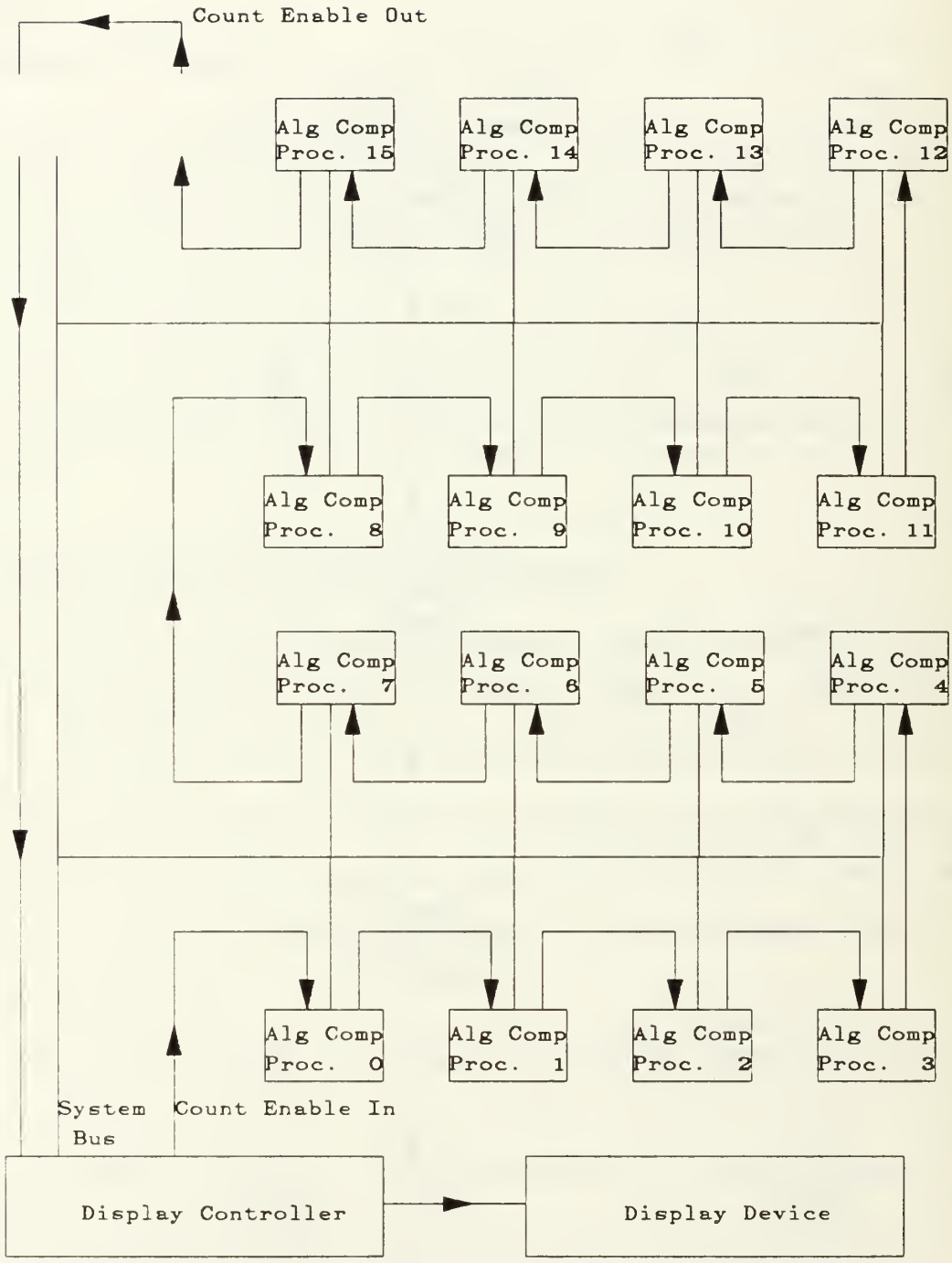

Figure 4.
Multiple Algorithm Component Processor Interconnections

The second operation, that of accepting a subgrid definition into a particular algorithm component processor, has implications for both the size of the RAM of the processor, and for the performance of its external communication mechanism. For that operation, the algorithm component processor needs to be able to recognize when a subgrid definition is addressed to it, and then needs to be able to store that information into its RAM. From [1], we find that the size of the input to the algorithm component processor is 24 bytes for a single 2 x 2 subgrid (Figure 5a). If we assume 32-bit transfers to the algorithm component processor, this is a total of 6 references per 2 x 2 for the input operation, requiring an equivalent amount of RAM storage. For the total number of 2 x 2 subgrids expected to reside in a single algorithm component processor, 240 subgrids, this means a requirement for 5760 bytes of RAM storage.

The third operation, that of placing the coordinates generated in a particular algorithm component processor onto the system bus, has implications similar to that of the input operation. For the output operation, the algorithm component processor needs to be able to recognize when it should deposit its coordinates onto the system bus, and needs to be able to provide RAM storage for those output coordinates beforehand. From [10], we know that the largest output that can be generated for a 2 x 2 subgrid is 6 coordinate and drawing instruction quadruples (78 bytes) (Figure 5b). If we count the value indicating the number of coordinates output, we need to perform 20 32-bit transfers for the output operation, and need to provide an equivalent amount of RAM storage.

The fourth operation, that of generating the contours for the 2 x 2 subgrid whose definition is held in the algorithm component processor, affects the size of all the memories in the algorithm component processor. The algorithm in [5] is based upon a data structure, the contouring tree, that is created for every 2 x 2 subgrid of the larger grid. A pre-order traversal procedure applied to each tree generates the coordinates and drawing instructions pertaining to the represented 2 x 2 subgrid for the selected contour level. If we use that algorithm, this means we need to provide space for the tree traversal list tables (2681 bytes), the algorithm component miscellaneous variables (45 bytes), and the code that performs the algorithm component computation (3080

Input Contribution

(1) 4 quantities for the grid values on the corners of the 2 x 2 subgrid (16 bytes).

(2) 2 values representing the lower lefthand coordinate of the 2 x 2 subgrid (2 bytes).

(3) 2 values representing the orthogonal coordinate and the orthogonal coordinate type (2 bytes).

(4) 1 value for the contour level (4 bytes).

Total = 24 bytes

Figure 5a
Specification for the Inputs to the Contour Surface Display Generator

Output Contribution

(1) 6(x,y,z,drawing_instruction) quadruples (78 bytes).

(2) Number of output coordinates (1 byte).

Total = 79 bytes

Figure 5b
Specification for the Outputs of the Contour Surface Display Generator

bytes). (A comprehensive listing of all the data required in the algorithm component computation can be found in reference [4], Figure 3.1.) The estimates for the input, output, tree traversal tables, and miscellaneous are derived directly from the data and data sizes required for the computation of the algorithm component. These values represent the space needed for registers, random-access and read-only memories. Rounding the microcode memory requirement, 3080 bytes, to a power of two, and assuming a horizontal microprogramming for the algorithm component processor, we find that a 1024 x 32-bit ROM is required. The tree traversal list memory requirement, 2681 bytes, becomes a 2048 x 16-bit ROM in the same fashion. The rest of the memory requirement for the algorithm component processor is shown in Figure 3 as a 2048 x 32-bit RAM, which is used to hold the subgrid definitions, the coordinates generated, and any temporaries required to compute the algorithm component. In [1], it is noted that the ROMs and RAMs specified are expected to consume the majority of the area on the VLSI chip.

2.2. Larger System of Multiple Algorithm Component Processors

The interconnection of the algorithm component processors in [1] is described with respect to the issues of how operations and data are communicated. Figure 4 contains a view of the proposed interconnection scheme for the algorithm component processors. In that figure, each processor is depicted as being connected to a system bus, and a serial control line called the count-enable line. As indicated in Figure 3, the system bus provides both data and instructions to the algorithm component processor. It also provides the pathway for data output back to the display controller. Not so clear in that figure is the function of the count-enable line. The count-enable line is a one bit control line that runs in a daisy-chain fashion from one algorithm component processor to the next. Its function is to provide a processor addressed capability for operations indicated to the larger system of processors. Its effect is to serialize the execution of processor addressed operations such as data input and output. This is accomplished in the following manner. Each algorithm component processor uses the logical OR of the global control line contained in the system bus and the count-enable line to determine if it should gate in the instruction currently presented on the system bus. A signal on the global control line indicates a global

operation, and means that all processors of the system should perform the specified operation. Global operations are used to initiate the highly parallel computations, i.e. the algorithm component computations. A signal on the count-enable in line for an algorithm component processor indicates a processor addressed operation, and means that the instruction and any following data on the system bus are addressed to that specific processor. Once an algorithm component processor has gated in a processor addressed instruction and its data, it then sets the count-enable out line high. The setting of the count-enable out line to high indicates to the next processor in the chain that it should gate in the instruction and data next on the system bus.

One note on the interconnection scheme in [1] is that the output is designated for a single 32-bit wide pathway into a display controller. The reason for this limitation is that the currently available display devices to which the output is directed, only have a single such pathway for display list modification.

Once the interconnections are specified in [1], the widths of (1) the system bus data and control lines, (2) the count-enable lines, (3) the external instruction register, and (4) the external data register are estimated. The system bus and count-enable lines sizes are the most important because they extend across VLSI chip boundaries, and hence require package pins. The count-enable lines require two bits, one into and one out of each algorithm component processor. This requires two pins on the VLSI chip. The system bus specification is more difficult in that we have both data and control line widths to specify. The width of the data portion of the system bus is chosen to be 32 bits. This figure is based upon the number of pins we expect to be able to spare on the VLSI chip, and upon the fact that a 32-bit processor is assumed. The control portion of the system bus has the following components: (1) global/processor addressed bit (1 bit), (2) instruction bits (3 bits), (3) data transfer control lines (6 bits), and (4) miscellaneous control lines (6 bits). The sizes indicated for the data transfer and miscellaneous control lines are taken from the bus designs for similarly sized processors and are not exact. The values quoted only serve as an estimate on the number of control signals expected. Consequently, the total estimate for the control portion of the system bus is 16 bits for a bus total of 48 bits. Adding the two pins

for the count-enable lines, this means a minimum of 50 pins on the VLSI chip. This is considerably under the 114 pins seen on the Motorola MC68020, and allows room for additional pin requirements [19].

The final size specifications made with respect to the system's communication mechanism are those of the external instruction register, and the external data register. The external instruction register needs only three bits, based upon the fact that there are only four operations signaled to the algorithm component processor. The purpose of the external instruction register is to hold a signaled instruction until the control portion of the algorithm component processor is finished with its previous operation and ready to execute a new one. The external data register is used to transfer data to/from the algorithm component processor from/to the data portion of the system bus. It is 32 bits wide to match the data width of the system bus. (Note: The width of the data bus discussed later in this thesis is 16 bits but its implementation in 32 bits can easily be applied when the display processing technology advances to the point of receiving a 32 bit data stream.)

The final architectural specification of [1] is an actual count of the total number of algorithm component processors required by the specified application in order to provide one-thirtieth of a second display generation and delivery. After some architectural modeling, it is stated that some 1000 processors are needed, each processor containing some 76 2 x 2 subgrids. A VLSI feasibility determination is carried out for the algorithm component processor with the finding being that each processor requires about 215K transistors. It is noted at that point that two million transistor VLSI chips are already being produced in the research lab [13], with ten million transistor VLSI chips promised in the time period ranging from the year 1985 to the year 2001 [14]. For the 1000 processors needed for the contour surface display generator, this means a total system size in the range of 112 to 21 VLSI chips.

Once the architectural specification and feasibility determination of the real-time contour surface display generator are established in [1] the paper ends with note on a problem with the design. This problem concerns the details of how the real-time contour surface display generator is interfaced to a display system. This is stated in [1] in a note on the output data rate of the

contour surface display generator. In the multiple processor interconnection picture, Figure 4, the output is shown to be destined for a display device, with that output passing through a display controller. The assumption for that data transfer is that it is accomplished via a DMA transfer mechanism of 32 bits width similar in operation to that of the DEC Unibus. Assuming that the output display is of average size, 89,100 32-bit memory references, this is a data rate of 10.7 megabytes per second. The delivery of data to the display system at the rate of 10.7 megabytes per second is somewhat faster than current display system technology allows. Compounded with this problem, is the fact that besides being able to deliver the picture within the given time constraints, it is necessary to maintain the functionality of the display system. This means that the addition of the contour surface display generator should leave intact the display system's capability for real-time display rotation, scaling. translation, clipping, and other assorted, real-time operations. The full specification of the architectural changes required for the display system by the contour surface display generator are not covered in [1].

2.3. Problems with the Original Architecture

One of the key problems with the architecture of the contour surface display generator of [1] is that its systems interface is not specified explicitly. A description is given in [1] of the functionality of the contour surface display generator with respect to its system interface but very little actual detail. In order to build this system, we need to provide further detail and resolve some serious system issues.

The first issue we need to address is that of system control. Given that the contour surface display generator requires 1000 processors, a clearer explanation needs to be provided for their overall control. In the original design, the control lines for the large system of algorithm component processors emanated from the Display Processing Unit (DPU) via the System Bus. The System Bus, besides being the control bus, also contained the data and address bus for the contour surface display generator. This is not the best arrangement for the data communication and control of the contour surface display generator. There are two problems with this design. The first problem is that the DPU is already quite busy controlling the display system. Adding the

functionality necessary to control the contour surface display generator is an excessive burden for the DPU.

The second problem is that the addition of the contour surface display generator to the DPU requires an extensive modification to the design of the DPU itself. As an intermediate stage to this larger effort, we instead have designed a self-sufficient system with a clearly defined standard interface that can be plugged into an existing display system. The interface we have chosen for this implementation is the IEEE Multibus interface [15]. The reason behind this choice is the prevalence of this type of interface on currently produced high-performance graphics workstations, one example being the Silicon Graphics, Inc. IRIS series [17].

The decision to have the contour surface display generator plug into the Multibus means that we must modify the design of [1], and add a controller to handle the functions previously designated as being handled by the DPU. This, in turn, means that we must reexamine the input and output operations to the system as originally proposed. In that system, a single bus handled both the input and output operations to the algorithm component processors to the exclusion of their concurrent operation. The reason cited at the time of the original design was for purposes of package pin count limitations. Technology changes that have occurred since that time have changed the upper bound on the number of pins available on a single VLSI chip. Consequently, our design eliminates this restriction and assumes that the input into the contour surface display generator is from the Multibus, and that the output from the system is by way of a separate, one-way 16-bit private data bus. (Note: The IRIS system already has such a bus into its Geometry Engines from the MC68010.) The implementation of the dual bus configuration is discussed in more detail below.

Another set of design decisions that was left out of the original architecture of [1] is the exact specification of the control signals sent to the contour surface display generator. In the original paper, the four operations to be carried out by the algorithm component processors were specified (reset, load subgrids, generate a new picture from a new contour level, and place the coordinates and drawing instructions generated onto the output bus), and then a very sketchy

look at the control lines necessary to handle those operations was performed. An operation which was not even considered was the need for some testing of the hardware prior to the system being run. We trace the specifics of how these operations are handled by way of the necessary data flow below in section 3.

Another set of control lines that need to be specified in more detail are those of the algorithm component processor's addressing mechanism. In [1], the count-enable lines provided an addressing method for inputing grid information into, and outputing coordinate data from the algorithm component processors. The design did not specify the details of what other control lines were to be used with the count-enable lines in order to coordinate the input and output operations. In the next section, we rectify this omission and formally specify the systems data flow, and lines necessary to sequence the correct flow of data for the system of algorithm component processors.

The only other control line specified in [1] was the global/addressed operation control line. The function of that line was to differentiate between a global and addressed function to the system of processors. In our design, this line has a different functionality. The remainder of the lines comprising the control to the system of processors are likewise not specified in [1], and are clarified and/or transformed below.

The final part of the design of [1] that needs to be clarified is that of the algorithm component processor. The algorithm component processor is depicted as a general purpose processor of the Motorola MC68000 class. The only architectural specifications made for that processor were with respect to the size of the memories required, and the input/output operations capabilities required. The major problem with the algorithm component processor as described in [1] is that it did not allow for the concurrent processing of data with the transfer of data to/from the processor. The system described below includes this capability.

As a side issue, the size of the input for the loading of subgrid definitions was set at 24 bytes in [1]. This number changes with the definitions within our system. The input size is reduced to 20 bytes because a global contour level is provided separately from the delivery of the subgrid

definitions to the algorithm component processors.

3. A Systems Design

The contour surface display generator is designed to be interfaced to an existing graphics system. This means that it is self-sufficient except for its I/O interface. The graphics system that is the target for the contour surface display generator is the Silicon Graphics, Inc. IRIS (see Figure 6). This system is chosen as target because currently, it is the highest performance graphics system that fits the requirements of the chosen application. This immensely powerful graphics system is integrated into a complete package via the IEEE Multibus. The contour surface display generator is designed to be plugged into that Multibus, and hence, connected into the IRIS system with very little modification to the IRIS. Because of this reliance on an existing system for which engineering drawings are available, a great number of the interfaces for the contour surface display generator are defined and need not be reworked.

In order to achieve the objective of contour surface display generator self-sufficiency, we define a control mechanism for the system of algorithm component processors. This mechanism is specified in terms of the devices which internally control the multiprocessing system. We define those devices in terms of their data flow characteristics and their hardware implementation.

The presentation of a data flow model for system behavior best illustrates how the contour surface display generator functions (Figure 7). The data flow model of Figure 7 is drawn as a state transition diagram, with the goal that once we define the data flow precisely, hardware realization can be accomplished directly from that information.

3.1. Algorithm Component Processors' Data Flow

The flow of data for the contour surface display generator is quite simple. The contour surface display generator only has four states during which data flows through the system (Figure 7). These states are (1) reset, (2) test, (3) load subgrid definitions, and (4) compute contours.

The reset state corresponds to the placing of the algorithm component processors into a known initial state. This is a global command to the system of algorithm component processors,

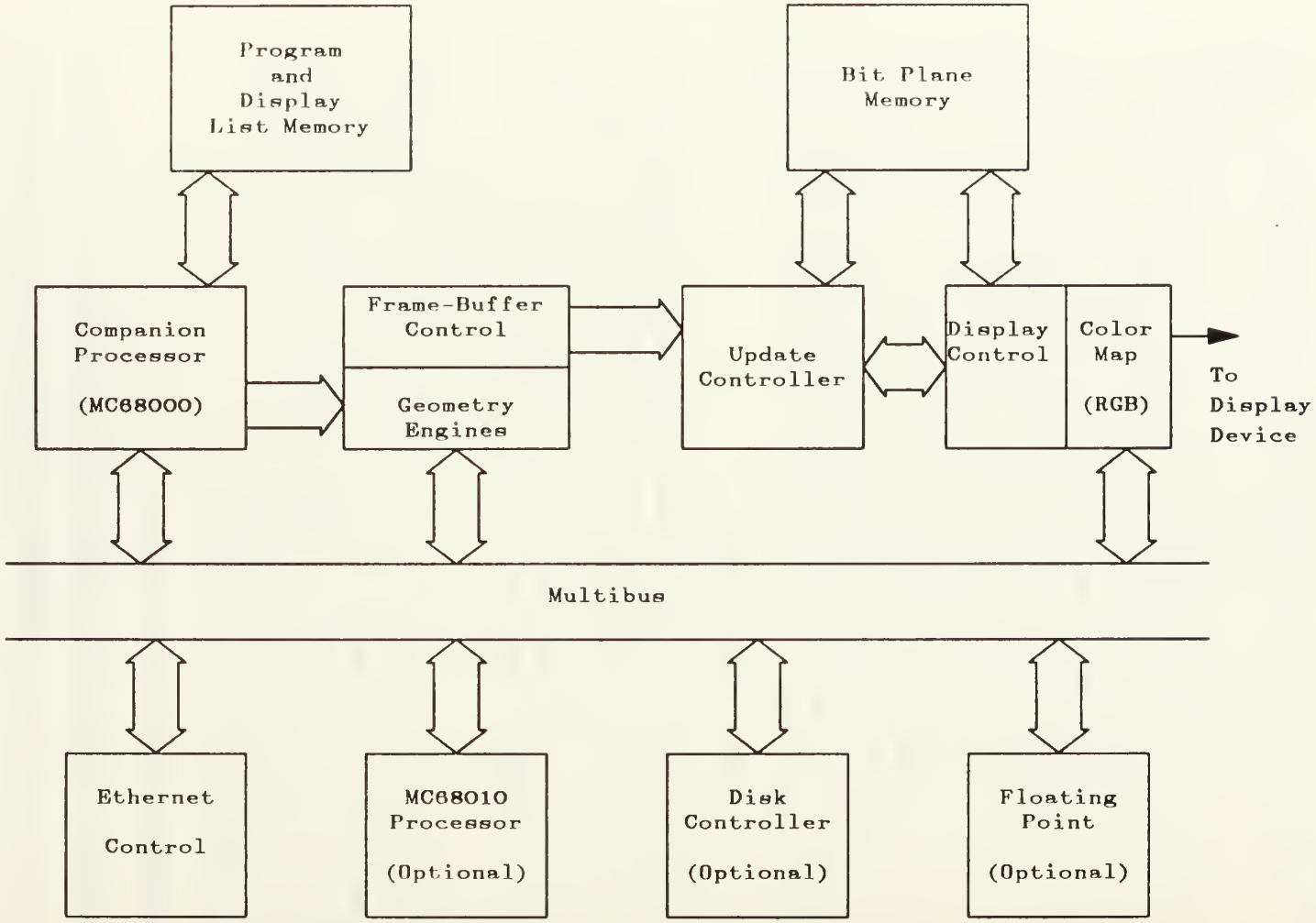

Figure 6. Silicon Graphics, Inc. IRIS System.

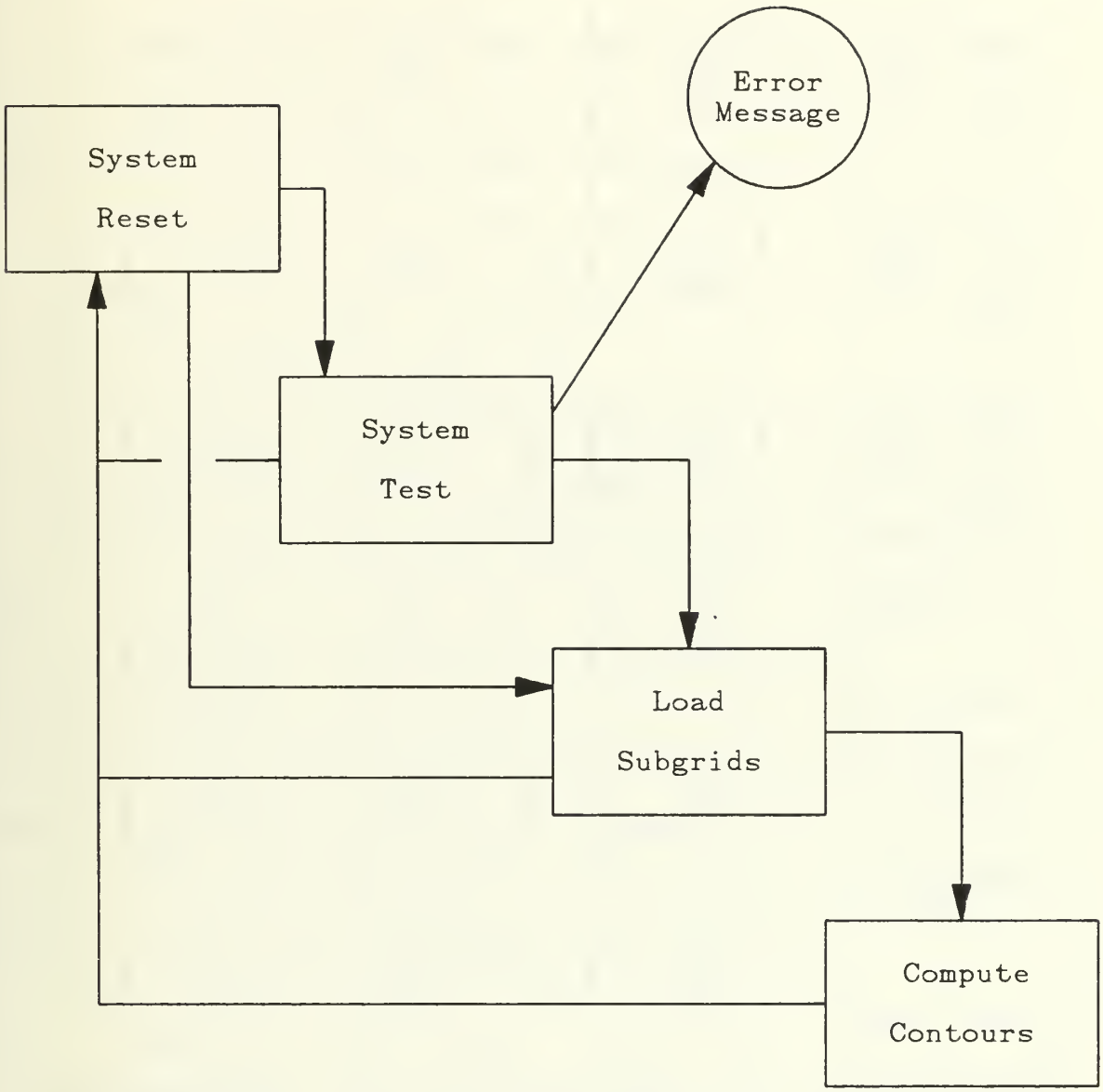

Figure 7. Contour Display System State Transition Diagram

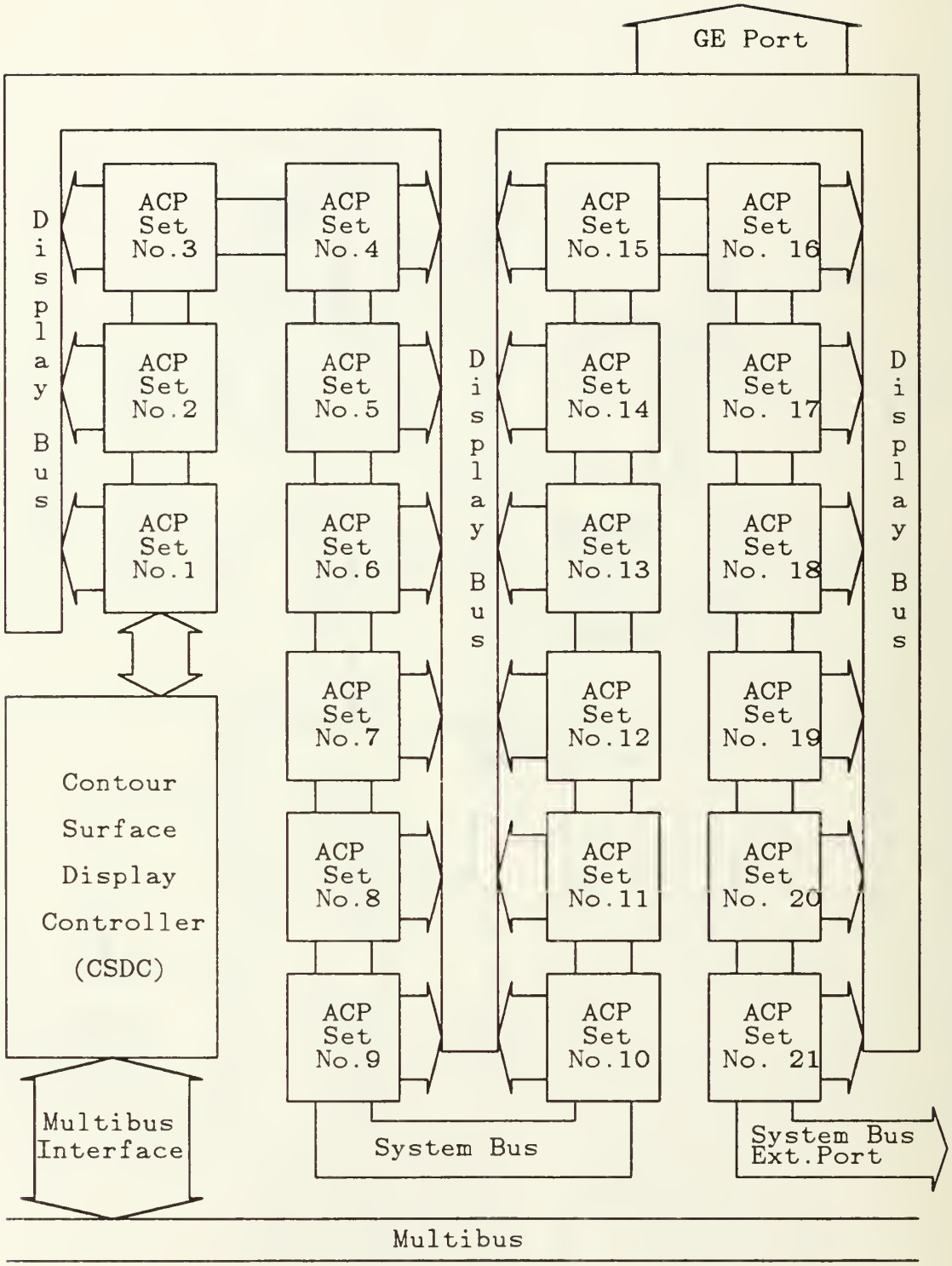

Figure 8. Contour Surface Display Generator.

and is accomplished concurrently for the entire system of processors. The reset command is generally given when the applications program using the contour surface display generator begins execution.

The testing state immediately follows the reset state. It has the purpose of conducting diagnostics on each algorithm component processor to ensure that no serious faults exist in the contour surface display generator. This state is explained in greater detail below.

The load subgrid definitions state corresponds to the mechanism described above for loading the 2 x 2 subgrids into the algorithm component processors. This state requires the capability for individually addressable algorithm component processors.

The fourth state for the contour surface display generator is that of computing the contours. In this state, the subgrid definitions are contoured according to an input contour level, with the results of that computation being supplied to the display processing unit of the target graphics system.

3.1.1. Reset Data Flow

Examining the state diagram from left to right, the first state we see is reset. When the applications program starts the system, the reset state is automatically invoked, and all algorithm component processors are initialized. This state can be enacted at any stage of systems operation. Reset always returns the contour surface display generator to the initial state.

3.1.2. Test Data Flow

After system initialization, there are two possible states the system can enter. The system can enter the test state, or the system can begin loading the subgrid definitions. If the testing path is taken, a signal is broadcast to all algorithm component processors that indicates that each should begin the on board test routines (on board chip test routines are a common feature in modern VLSI processor design). Once the global command for test has been issued to the system of algorithm component processors, the controller for the system can then cycle through that system of processors utilizing the count-enable line, checking to see if each processor has finished its

test routine. If an error occurs during the test state, a signal is sent to the controller signifying that an error condition exists. Operations are then suspended awaiting the rectification of the error condition.

3.1.3. Load Subgrids Data Flow

The load subgrid definitions state can be entered immediately after the reset state, i.e. without having to enter the testing state. The load subgrid definitions state requires a great deal of coordination between the applications program and the sequential loading process. The applications program has a large number of subgrid definitions it needs to distribute to the algorithm component processors. During this state, each subgrid is assigned to a particular algorithm component processor by a parent processor (MC68000/MC68010). These subgrid definitions are sent to the contour surface display generator one at a time until all 75,690 subgrids have been loaded. (Note: The load and distribution strategies for the subgrid definitions are discussed in section 5.)

The steps necessary for handling the loading of subgrid definitions are straightforward. First the algorithm component processors are configured to the load state (via a control signal). This defines the functionality of certain necessary response signals (count enable, write enable and wait). The count enable signal cascades through each algorithm component processor signifying a write operation to the selected processor. While setting up to receive data, the wait signal is sent to the controller to indicate that the algorithm component processor is setting up to receive its data. When the wait is removed, the write enable signal is activated and the algorithm component processor performs the subgrid load operation. Upon completion of the load, the write enable signal is removed and the count enable signal is forwarded to the next algorithm component processor in the array. The loading process continues until all subgrid data has been loaded and the count enable line is returned to the controller.

3.1.4. Compute Contours Data Flow

Once the contour surface display generator is loaded with subgrids, the final state of the state diagram, the compute contours state, can be entered. The response signals that are involved

in the compute contours state are the input count enable, output count enable, write enable, wait, transfer request, and transfer acknowledge signals.

The count enable signals function as an address bus for the contour surface display generator. It not only gives the system an efficient means to address each algorithm component processor but it also acts as an arbiter for accessing the system's buses. This alleviates all the complex circuitry required in traditional multiprocessing systems for the addressing and arbitration functions.

Two count enable signals are necessary to optimize the rate of system input and output. This involves the implementation of two separately defined signals, one for input and one for output. The input count enable signal originates from the system controller and cascades through the algorithm component processors when a contour level is placed on the system data bus. The write enable and wait signals are the coordinating signals which form the two signal handshake protocol necessary for the efficient transfer of data. The transfer of the contour level occurs when the algorithm component processor is ready to receive (wait line is inactive) and the controller is ready to send (write enable is active). Since each contour level is 4 bytes, this input requires few clock cycles and each new contour level is provided to all algorithm component processors in a virtually instantaneous fashion.

Once the contour level has been input to the algorithm component processor, computations can begin. The execution of each contour producing subgrid is determined by checking certain boundary conditions [6]. (Note: The details of this process are not delineated in this study but are merely taken into consideration when constructing the hardware needed to manage the operation of the algorithm component processor.) Once these computations begin, coordinates and drawing instructions are produced and staged for delivery to the display processing unit. The output of these coordinates and drawing instructions is controlled by the output count enable signal which cascades through the array of algorithm component processors to sequentially retrieve data for the output bus. The arrival of the output count enable signal at an algorithm component processor indicates that it is that processor's turn to transfer data. At this point, the algorithm

component processor must use two additional signals to coordinate the output of data to the display processing unit. The first signal is the transfer request signal which originates from the algorithm component processor and is sent to the target display processing unit. Its purpose is to indicate that data is available for processing and is awaiting the acknowledgement to begin its transfer. If the display processing unit is ready to receive this data, it sends an acknowledge signal back to the algorithm component processor. This return signal indicates to the system that the transfer is in progress.

The data transfer takes place in the following fashion. When the output count enable signal arrives at an algorithm component processor, the processor halts what it is doing, verifies whether or not there is output and initiates the appropriate action. If there is output, the appropriate action is to establish a link between the algorithm component processor and the display processing unit. Once established, the transfer takes place in the manner described above. Upon the completion of the transfer, the count enable signal is sent on to the next algorithm component processor and the cycle is repeated. If an algorithm component processor receives the output count enable signal and has no data to transfer, it merely sends the output count enable on to the next processor.

3.2. Internal Hardware Control

Once we have a description of the systems data flow and a sound general architectural design, the hardware configuration designed to meet these requirements can be formally specified [1, 8]. When the data flow was described earlier, a reference was made to a dual bus configuration which separated systems input from systems output. A dual bus configuration was chosen to maximize the amount of concurrency in the system due to the autonomous nature of the input with respect to the output. The data flow within the contour display generator is unidirectional in that contour levels go in on one bus and coordinates and drawing instructions come out on another. More importantly, however, is that the output bus is necessary in order to interface to the Silicon Graphics Private Bus. Figure 9 shows the buses and the data flow of the contour surface display generator.

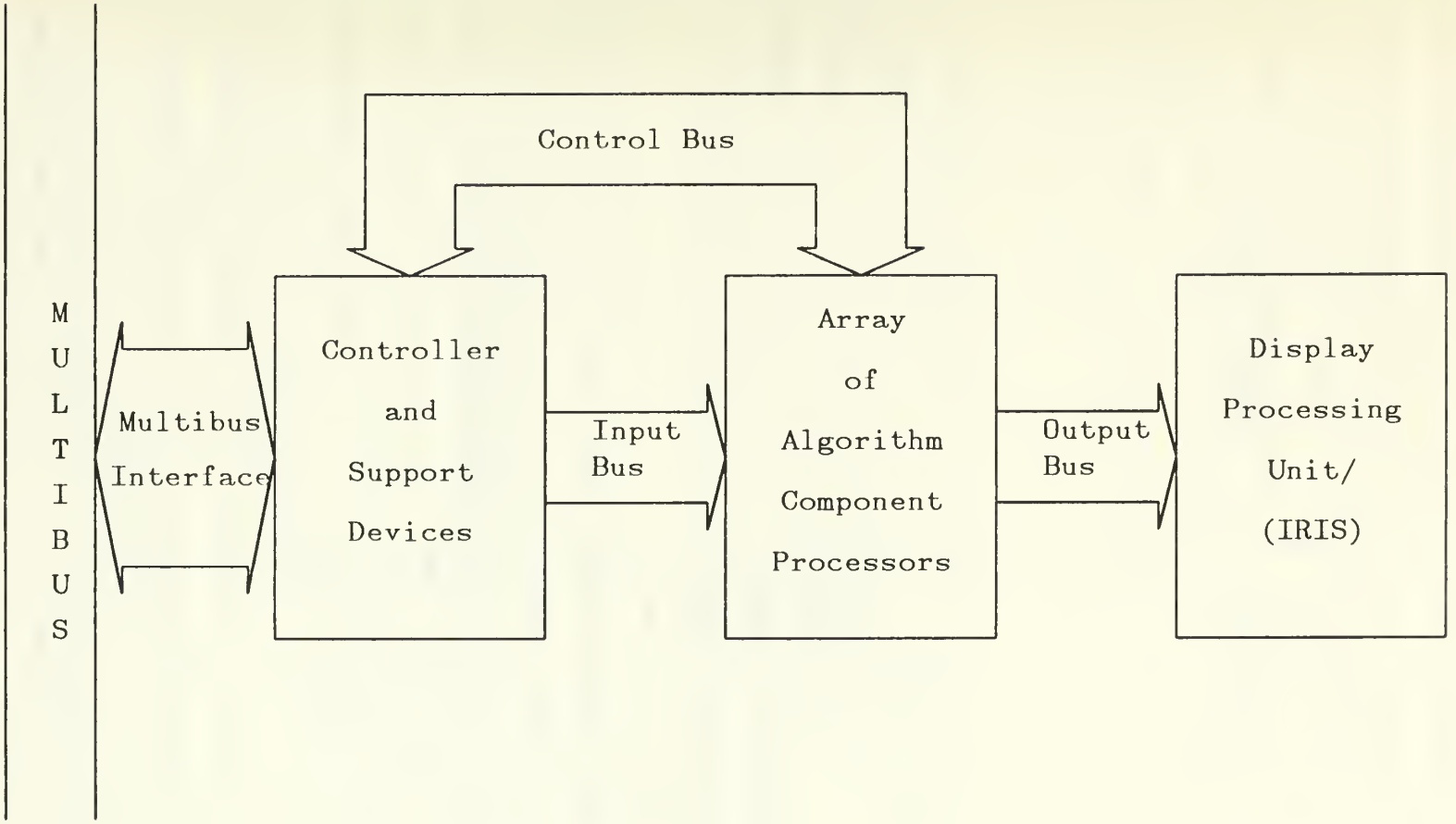

Figure 9.
Contour Surface Display Generator: Buses and Data Flow.

The input bus is the medium responsible for delivering subgrid definitions and contour levels to the array of algorithm component processors. Because this is the only data required to be transmitted on the bus, the bandwidth of the input bus does not need to be very high. The rate at which subgrid definitions are loaded into the algorithm component processors does not directly affect the real-time capabilities of the system. The real-time capabilities of the contour surface display generator are determined by the rate at which data can be produced in each algorithm component processor. This, in turn, directly affects the rate of output to the display processing unit. The output bus is responsible for delivering the coordinates and drawing instructions to the display processing unit.

The control bus for the contour surface display generator contains all the control lines necessary to manage the data flow on the input side of the system (see Figure 10a). Two additional control lines are required on the output side of the system to coordinate the two wire handshake between the algorithm component processors and the display processing unit (Geometry Engines). Figure 10b shows the signals that are needed for all the pin assignments of the algorithm component processor.

The control lines work in conjunction with the data flow described above. The control lines of the input bus consist of 4 lines for the count enable lines (input and output control), 1 line for test acknowledge/wait signals, 1 line for error indication, 1 line for write enable, 4 lines for setting the state configuration, 1 line for ground, 1 line for clock and 1 line for power. The output bus only requires the Geometry Engine request (ge.req) and the Geometry Engine acknowledge (ge.ack) lines for its control. The output bus is discussed in more detail below. The ground, clock and power lines are all standard control lines and need no explanation. The other lines of the input bus, however, do warrant brief explanation since they are a bit more complex in functionality.

The simplest way to describe the functions of the control lines is to relate them to the data flow described above. The timing diagrams provide a general understanding of how the systems functions are achieved (Figures 11-14). These diagrams are related directly to the system state

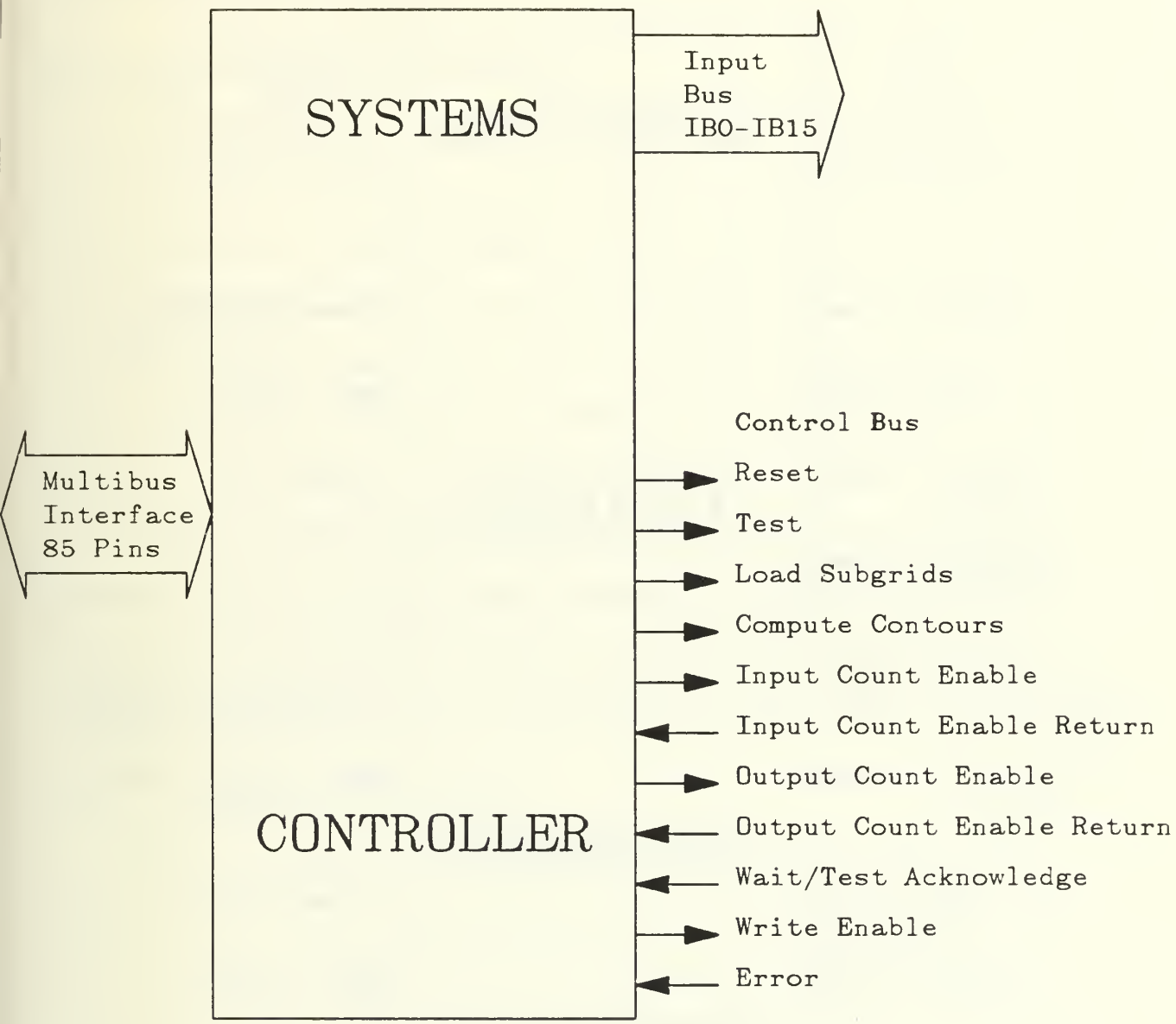

Figure 10a
Functional Pin Diagram of the Systems Controller

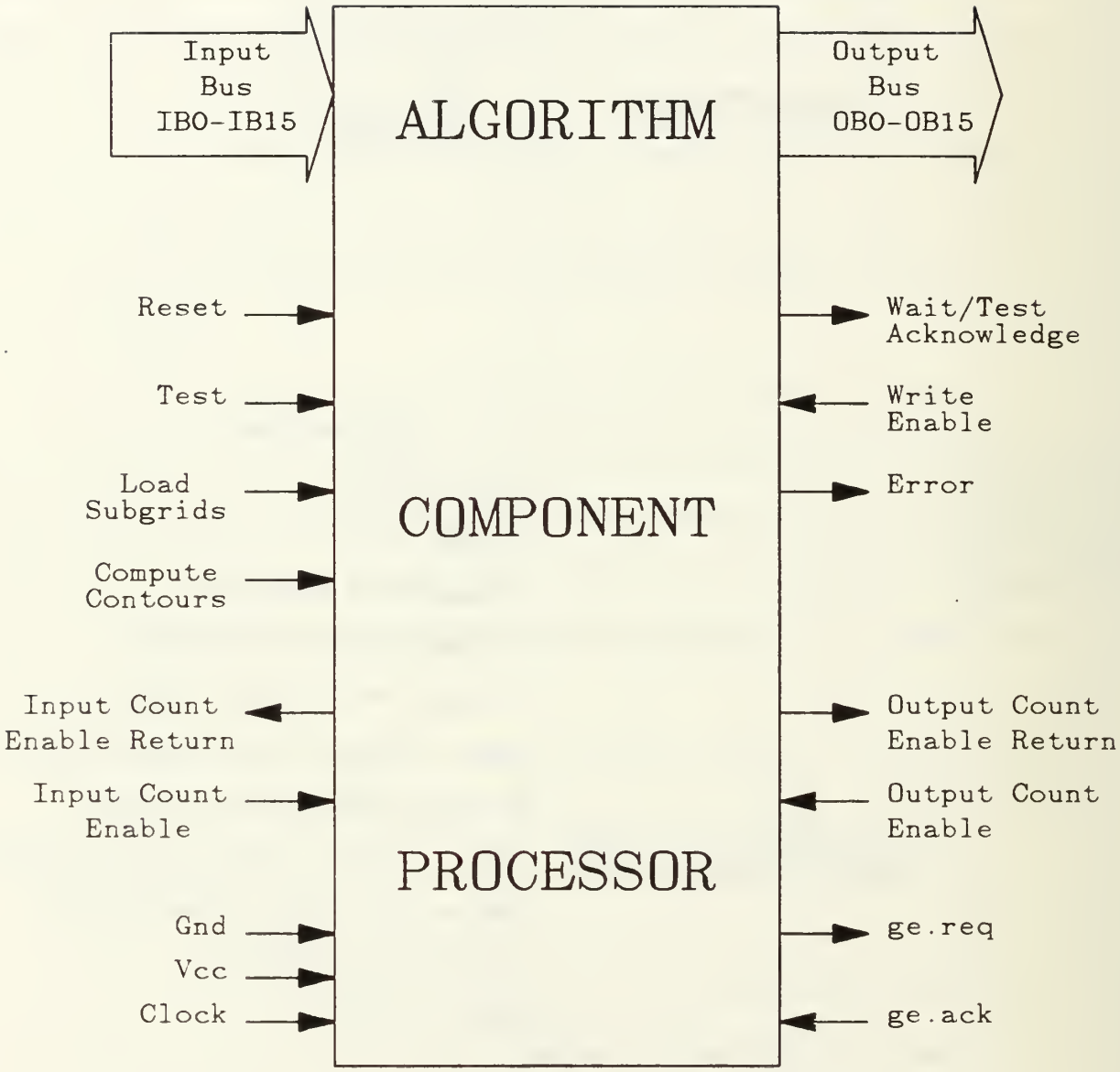

Figure 10b
Functional Pin Diagram of the Algorithm Component Processor

diagram and assist in a broader understanding of system behavior (Figure 7). The principles used in the explanation of control signals closely resemble those used for the timing of the control signals of the Z80 microprocessor [16].

3.2.1. Reset Control and System Initialization

The four lines which control the states of the contour surface display generator are the reset, test, load subgrids, and compute contours lines. The reset line requires no timing diagram since its only purpose is to set the algorithm component processors to an initial state. A reset can occur at any point during the operation of the contour surface display generator and therefore can preempt any other activity in the system. Even though this line needs no particular timing, it still flows through the controller so that the controller can properly configure the system to the initial state.

The other three system control lines put the system into the appropriate state configuration as described earlier. For clarity, they require timing diagrams to demonstrate their functional implementation. In order to remain consistent with our explanation of these timing diagrams, we assume all of our control lines are low when inactive and high when active.

3.2.2. Test Control and Error Conditions

The first state in the state diagram after the reset command is the test state. Figure 11 describes the active control lines for the timing of this state. When the test control line goes high, all the algorithm component processors are immediately placed in the test state. Each processor activates an on board test routine and upon completion signals an internal status register that the test has been either successful or unsuccessful. After the test control has been set high, the controller sends the count enable line to the first algorithm component processor. The count enable line cascades through the array of algorithm component processors verifying that the test has been successful for each processor. If the test is successful, the algorithm component processor which contains the high count enable line sends the test acknowledgement line high to the controller. This enables the controller to keep track of the whereabouts of the count enable line as it

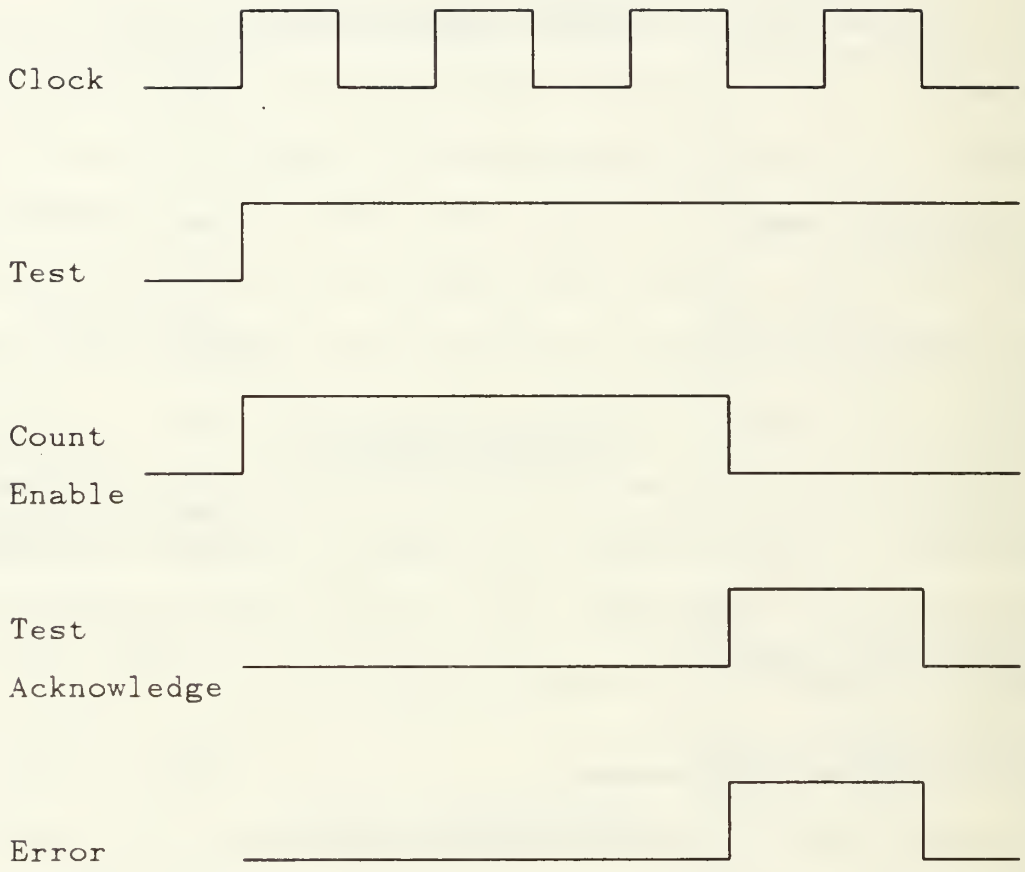

Figure 11.
Test Timing Diagram for the
Contour Surface Display Generator

cascades through the array of algorithm component processors.

If an error does occur, the error line signals to the controller that an error condition exists. When the controller sees the error line go high, all further operations are suspended pending rectification of the error. The design of the contour surface display generator does not currently include tolerance to such errors. A discussion of this topic is included in section 6.

3.2.3. Input of Subgrids Control

When the testing module is successfully completed (count enable line returned to controller) or the applications program signals that a test is unnecessary, the load subgrids control line is sent high by the controller. This signifies the beginning of the loading of the subgrid definitions and configures the algorithm component processors to allow open access of their RAM memory to the systems controller. This simplifies the loading process for the controller since it is essentially a write to memory operation and can be done quite efficiently.

The load subgrids process only requires the movement of 20 bytes of data for each 2 x 2 subgrid. Figure 12 shows the timing diagram for the load of a single subgrid. (The loading of subgrids can easily be extended to load the maximum amount needed for a particular algorithm component processor. The only factor which affects the amount sent at one time is the circuitry and RAM capabilities of the algorithm component processor.)

The count enable line acts as the address mechanism for the memory write cycle of the load operation and signifies which algorithm component processor is to be loaded. Each algorithm component processor has hardware that determines the absolute location where each subgrid is loaded. We call this hardware device the load pointer.

When the count enable line is set high in an algorithm component processor, the load pointer within the processor is activated. The load pointer points to the absolute location of the first available byte of RAM for the subgrid data. Concurrently, a wait signal is delivered to the controller signifying that the processor is not yet ready to receive its data. Once the pointer is set, the wait line is removed, the write enable line is set high and the transfer begins. Once the

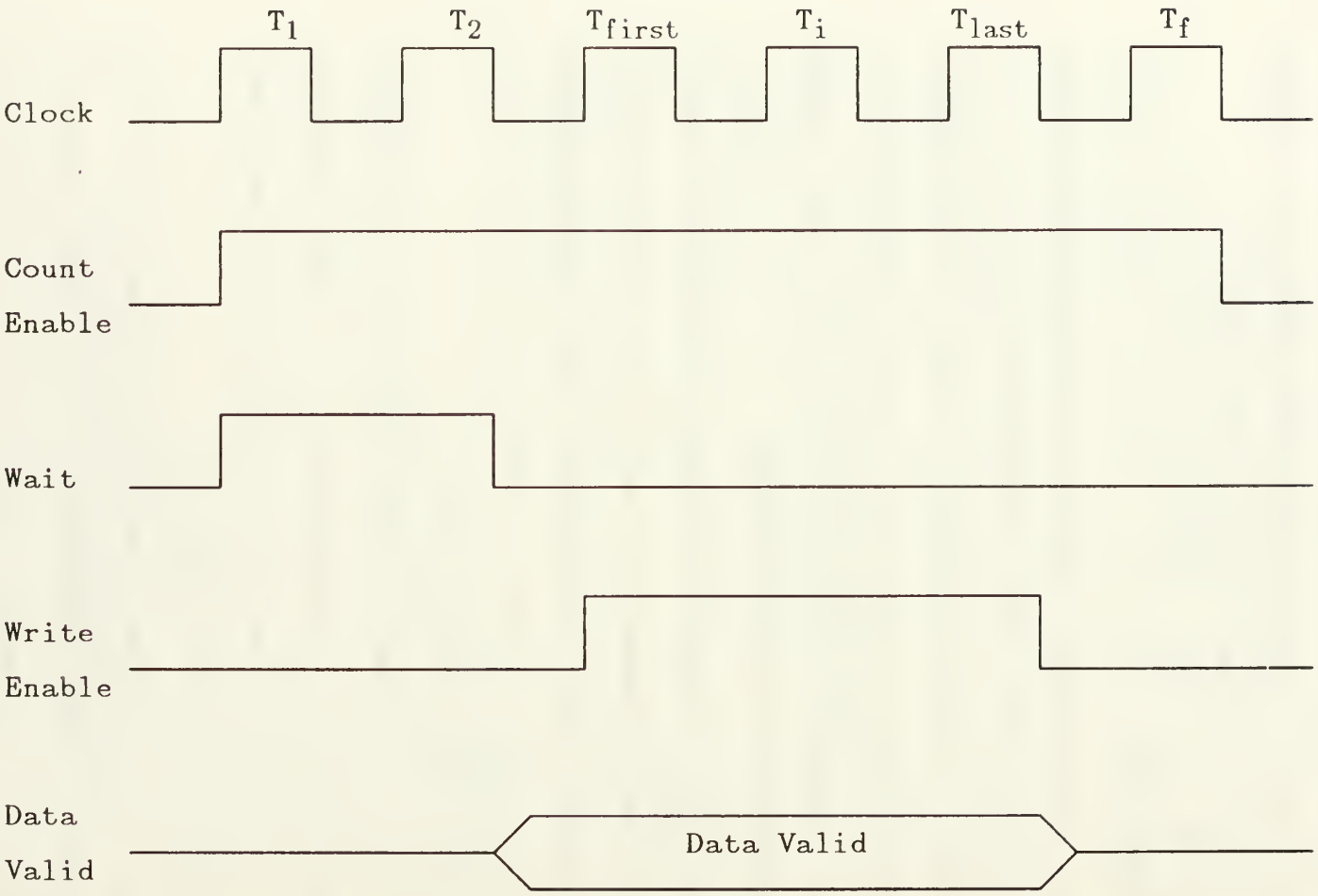

Figure 12.
Timing Diagram for the Load Subgrids State of the Contour Surface Display Generator

controller has sent all the subgrids destined for that algorithm component processor, the write enable line is set low signifying the end of the transfer.

The transfer of the subgrid definitions is very efficient because the address for the data being transmitted is known ahead of time and the controller has some knowledge as to how much data each of the algorithm component processors can hold. The total number of subgrids to be loaded for any three-dimensional surface display is 75,690 [1]. These subgrids are distributed equally to the 50 algorithm component processors that comprise the system. Each algorithm component processor receives roughly 1514 subgrids. (The derivation of this value is covered in detail in section 5). When a processor receives its 1514 subgrid definitions, it forwards the count enable line to its successor processor and the cycle is reiterated until all the algorithm component processors are loaded.

3.2.4. Compute Contours Control

The final part of the state diagram is the compute contours state (Figure 7). This is probably the most demanding state in terms of resource management. This state is responsible for concurrently performing three different tasks: the loading of contour levels, the computation of the contours and the output of those contours in the form of coordinates and drawing instructions. All of these functions are continuous in nature and contribute to the real-time capabilities of the system. Since these functions are independent from each other and use separate communication media, their control is more complex than the other states that have been discussed.

The compute contours input phase begins when the compute contours control line is placed high. This signifies to the algorithm component processors that they are now operating in the compute contours state configuration. Entry into this state puts the algorithm component processors into a suspended state, awaiting the arrival of the input count enable line. The input count enable line serves as an interrupt line at the beginning of the input process. Figure 13 shows how the response lines are interrelated. The interrupt of the algorithm component processor by the input count-enable line suspends all processing in that processor. At that time, an interrupt routine which services the input of the contour level is entered. Coordination of the data transfer is

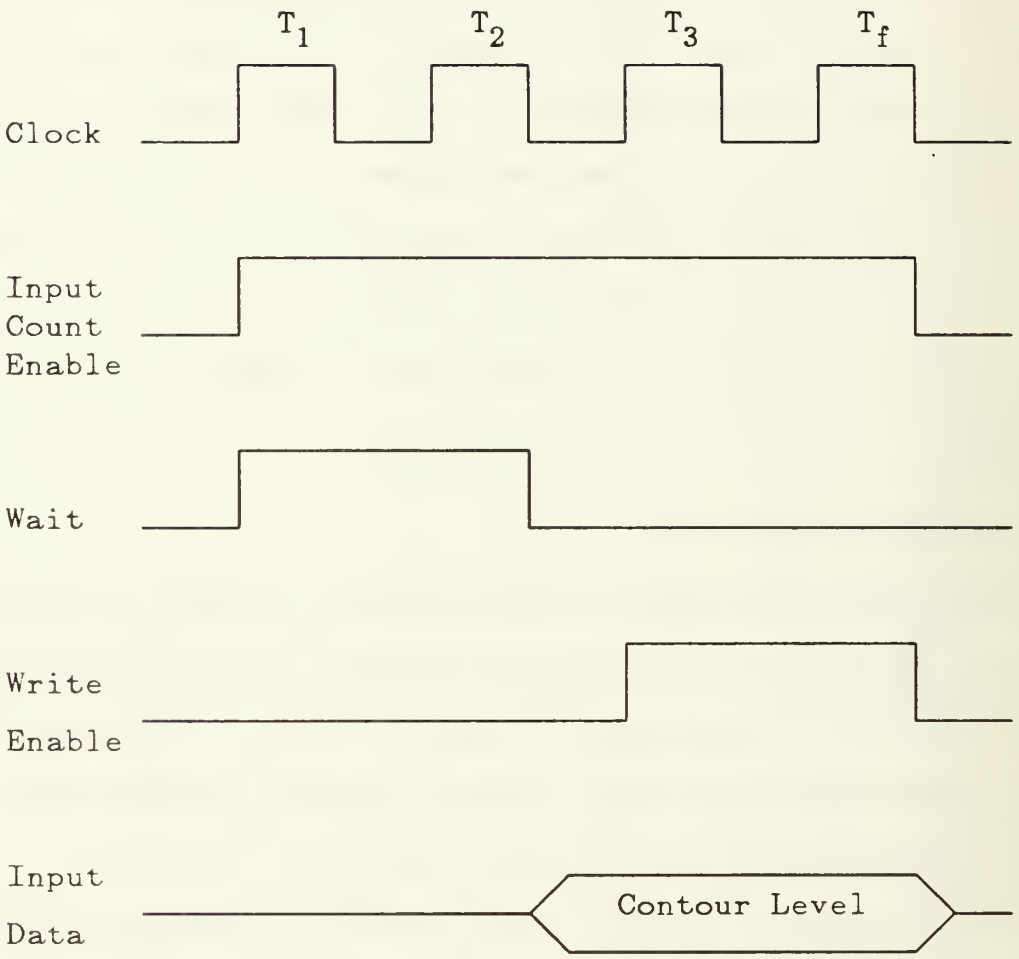

Figure 13.
Timing Diagram for the Input of a Contour Level for the Contour Surface Display Generator

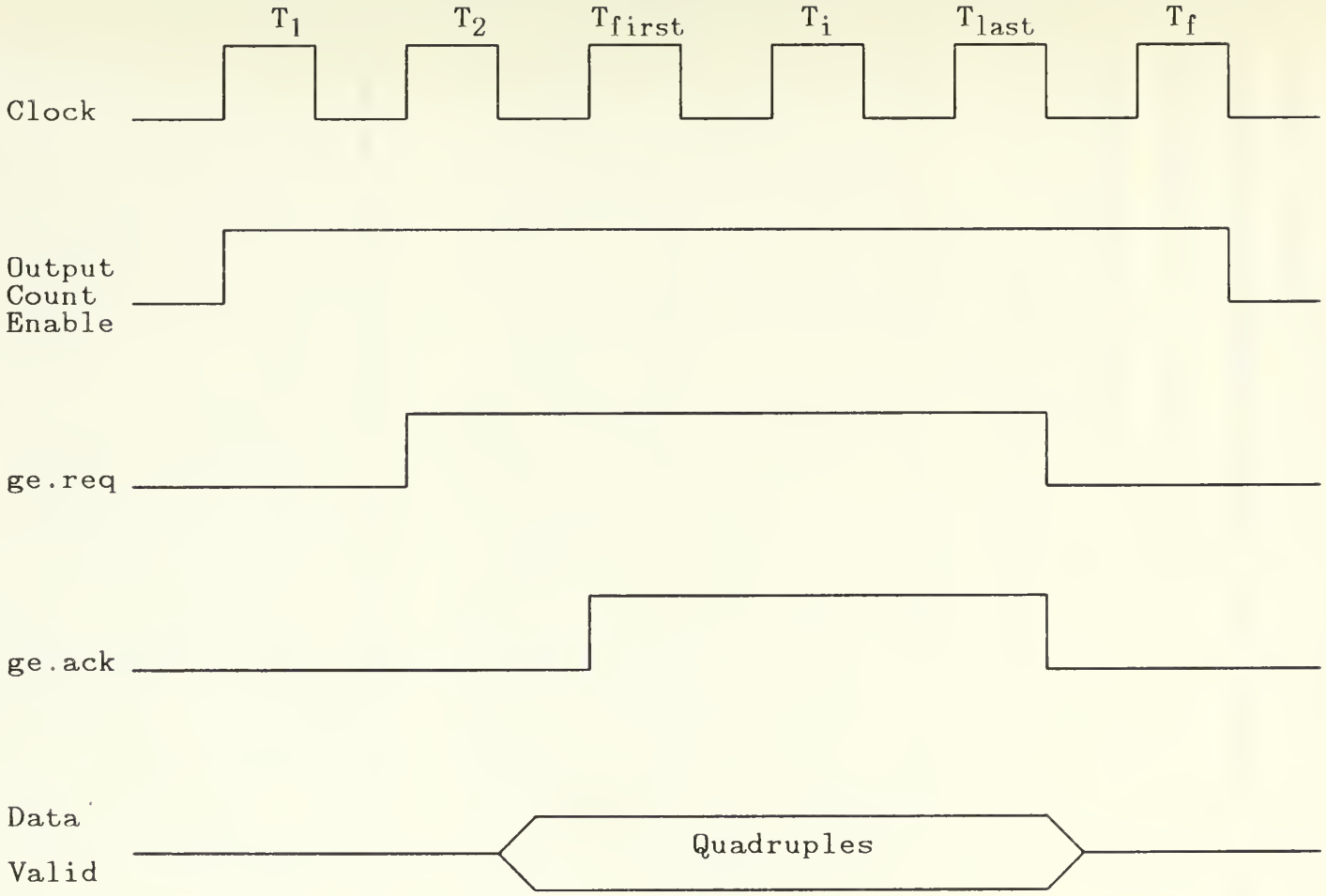

Figure 14.
Timing Diagram for the Output of Coordinate and Drawing Instruction Quadruples for the Contour Surface Display Generator.

done using the write enable and wait lines. Upon entering the interrupt the algorithm component processor forces the wait line high informing the controller that it is setting up to receive data. When the wait line is set low, the controller knows that the algorithm component processor is ready to receive the data. The controller immediately sets the write enable high while concurrently placing valid data on the input data bus. The data is then transferred, which completes one load contour level cycle. The input count enable line is passed on to its successor processor and the cycle is reiterated. The algorithm component processor which received the contour level performs a clear operation on any operations in progress. Immediately following the clear, calculations leading to the output of the coordinates and drawing instructions begin.

Since the output of coordinates and drawing instructions is independent from the input of the contour level. both processes can occur in parallel. The synchronization of the output is done by another count enable line which is continuously cycled through the algorithm component processors in search of data to be output to the display device. The volume of output resident in each algorithm component processor's output staging area is dependent directly on the turn around time of the output count enable line. The length of time it takes for the output count enable line to return to a particular processor determines how many computations are completed in that time. The display processing unit only sees a continuous stream of coordinates and drawing instructions.

The only portion of the output process involving the controller is the output count enable signal. The controller gives this signal a place to start and a place to end. Once the compute contours state is entered. the output count enable line is activated by the controller and is reactivated each time it returns from the last algorithm component processor in the array. When the output count enable line arrives at an algorithm component processor. it interrupts that processor. The processor then delivers any data destined for the display device to the output bus. The algorithm component processor either initiates the transfer of data or passes on the output count enable line to its successor processor (Figure 14). If there is data to transfer, the transfer request (ge.req) is sent to the display device. If the display device is ready to receive the data, it replies

with an acknowledge signal (ge.ack). The transfer then begins in a burst DMA type fashion. When the algorithm component processor runs out of data to send, it sets its acknowledge signal low to signify the end of data. The transfer finishes when the display device sets the acknowledge signal low. If at any time, the display device sets the acknowledge low, the transfer is suspended until the acknowledge is again set high. The transfer of the concerned data is then resent to ensure its correct arrival.

4. Systems Components and Interfaces

The lines described in the section above explicitly define control aspects of the contour surface display generator. Still to be defined are the components and interfaces necessary to make those control lines work. The components that make this system work are the systems controller and the array of algorithm component processors. The algorithm component processors are fully independent processors. The systems controller, on the other hand, requires a variety of support components to enable it to function. The controller's support devices are described below. The algorithm component processor needs minor adjustments to the design that was specified in [7].

Once we have described the algorithm component processor and the systems controller in detail, we can then detail how the contour surface display generator interacts with the external graphics system. This interaction is accomplished via a Multibus interface and a Silicon Graphics, Inc. IRIS system Private Bus interface. The Multibus backplane contains all the printed circuit boards necessary to make the integration with a graphics workstation possible. It provides each board in the backplane with a means of communicating with the others. It enables the contour surface display generator board to be remotely controlled by a parent processor or bus master. This bus master determines when the contour surface display generator is used and supervises its initialization, loading, and delivery of contour levels. This interface is discussed below.

The Silicon Graphics, Inc. IRIS Private Bus is the output bus of the contour surface display generator and is not connected to the Multibus. Its data flow is unidirectional. The IRIS Private Bus is connected directly to the display processing unit (Geometry Engines). The interface requirement for this 16 bit data bus is discussed below.

4.1. Systems Controller

Control of the array of algorithm component processors involves the integration of several different components. The one which coordinates the operation of all other components is the systems controller. The systems controller converts incoming signals from the Multibus bus master into signals which make sense to the algorithm component processors. (Note: The Multibus bus master is the board in the Multibus Backplane which places the commands on the Multibus. A slave, on the other hand, reacts to these commands on the Multibus. The commands exchanged are discussed in section 4.3.) These signals were explained earlier by way of their data flow and actual control line implementation. To adequately describe these signals from the controller's point of view, a pseudocode algorithm is provided in Appendix A. This algorithm functionally describes what the controller must do when it receives commands from the Multibus bus master.

4.1.1. Controller Algorithm Description

The introduction of power to the Multibus automatically sends a reset command to the controller (INIT). Upon receipt of this command, the controller immediately sets the reset line high for the contour surface display generator. The controller then waits for a command from the Multibus bus master to indicate that the generator is going to be put to use. When the Multibus bus master addresses the contour surface display generator via the Multibus, the controller sends the bus master a message which inquires whether a test is desired before the loading of subgrids. If a test is desired, the bus master sends a message back to the controller indicating that a test should be performed. The controller then immediately sets the test line high and the testing process begins.

Once the testing process is initiated, the controller activates the count enable line. When that line returns to the controller the testing process is complete and the load subgrids process can begin. During the test operation, the controller keeps track of where the count enable is in the array of algorithm component processors. It does so by storing a counter value internally. Whenever a test acknowledge signal is returned from one of the algorithm component processors,

that counter is incremented by one. If an error occurs during this testing process, the controller knows exactly which algorithm component processor is responsible from this internal counter. Such errors are relayed to the bus master by the controller.

The indication of an error can occur in two different ways. The first of these is by direct indication. On each algorithm component processor is an error line. If that algorithm component processor has an error and its on board test routine detects it, the line is set high and the controller is directly notified of the error condition. The second way of detecting an error is indirectly. When the on board test routine fails and hangs up, an indefinite postponement of operations occurs. To prevent this, the controller contains a second counting device which is reset after each test acknowledge signal. It has a threshold value and if reached during one of the intervals between test acknowledgements, an error condition is indicated. If either one of these error conditions occurs, the controller immediately sets the test line low, sets the reset line high and notifies the bus master of the error condition.

If the bus master decides that testing is unnecessary, the loading of the subgrid definitions is begun. The controller starts this process by setting the load subgrids line high, thus configuring all the algorithm component processors in the load subgrids state. The controller then awaits delivery of the first batch of subgrids to the first algorithm component processor. The bus master sends these subgrids and the subgrids are loaded directly into the controller's local RAM memory. (Note: The bus master determines ahead of time which subgrids are to be loaded into which algorithm component processor. A distribution method which determines this is discussed in section 5.) The controller waits until all the subgrids for one algorithm component processor have been loaded into its local RAM before beginning the load cycle.

At the start of the load cycle, the controller sets the count enable line high (Figure 12). This forces the first algorithm component processor to prepare for the receipt of the subgrids. That algorithm component processor sets the wait line high which forces the controller to wait until the processor is ready to receive the subgrids. When the wait line goes low, the controller sets its write enable line high while concurrently placing valid data on the bus. At this point, the

transfer is formally in progress and continues until the write enable line is set low. This uninterrupted transfer of data is necessary for efficiency as well as predictability. The transfer of data within the contour surface display generator is synchronous in nature. The transfer of data over the Multibus is asynchronous. This is the main reason why all the subgrids are staged in the controller's RAM by the bus master prior to the subgrid load cycle beginning. This load process is continued for each and every algorithm component processor in the same fashion until completion. When the loading process is finished, the controller immediately configures the system to compute contours.

The controller configures the system to the compute contours state by setting the compute contours control line high. The controller then activates the input count enable line. The first contour level is then provided to each of the algorithm component processors in a serial, cascaded fashion. The controller awaits the arrival of this contour level from the bus master. When it is received, the controller initiates the input cycle of the compute contours state (Figure 13). If the wait signal of the first algorithm component processor is low, the transfer process can begin. The controller sets the write enable line high, transfers the contour level, and then pauses waiting to repeat the process. Upon receipt of the input count enable, the next algorithm component processor sets the wait line high signifying that it is setting up to receive the contour level. When the wait line is set low, the controller again sets the write enable line high and the cycle begins for the next algorithm component processor. This process is repeated for each algorithm component processor until the input count enable line is returned back to the controller.

Once the first contour level has been provided to the array of algorithm component processors, the output count enable line can be activated. The purpose of this line is to cascade through the algorithm component processor array in search of coordinates and drawing instructions ready for output. The reason this line is not activated earlier is that this line causes an interrupt to occur each time it is returned to the controller. If there are no coordinates and drawing instructions to be placed on the output bus, the line continually interrupts the controller during the load contour level operation. By waiting to activate the output count enable until after the first algo-

rithm component processor has been loaded, the system has a chance to get going before a continual interrupt by the output count enable overwhelms the controller. Since the output bus is driven directly by the algorithm component processors and the output count enable line supports the output bus the less time the output count enable line spends in the controller the better.

4.1.2. Controller Hardware

The hardware component necessary to implement the controller is diverse in functionality. Its functions include: (1) the interfacing to RAM and ROM memory, (2) the capability for burst DMA operations, (3) the capability for being programmed to manage the array of algorithm component processors, (4) the capability for background timing circuitry management and (5) the capability for interfacing directly with the Multibus Backplane. This type of integrated processing chip is hard to find in one complete package and as such may require the use of several different components. There are two chips that are available commercially which more than adequately fill the requirements, the Motorola 68230 Parallel Interface/Timer and the Motorola 68000 Processor. The integration of these two components is beyond the scope of this study. The use of the Motorola 68230 for the parallel interfaces of the system has one drawback; its data bandwidth is only 8 bits. The integration of this component into the contour surface display generator requires the development of a scheme to work two of these chips in parallel to achieve the desired 16 bit data path.

Possible enhancements to the architecture of the controller are addressed in section 5. Some of these enhancements take advantage of the tremendous capabilities of the two chips mentioned above. Other support components are needed for off board interfacing. The discussion of these requirements is included with the system's interfaces below.

4.2. Algorithm Component Processor

The component that is responsible for the production of the coordinates and drawing instructions for the contour surface display generator is the algorithm component processor. Each of these processors is identical and functions independently in the production of the outputs. A

proposed architecture has already been developed for the algorithm component processor and requires only minor modification (see Figure 3 and [1, 5-10]). Before specifying these modifications, however, an algorithmic description of the processor's functionality must be given to substantiate any changes that are required in that original architecture.

4.2.1. Description of the Algorithm

The introduction of power to the Multibus automatically supplies the power required by the algorithm component processor. The algorithm component processor does nothing until it is given a signal from the systems controller. The first signal that the controller sends to the algorithm component processor is the reset command. Upon receipt of this command, the processor initializes all of its memory and registers to zeroes and then waits until the next command is received from the systems controller.

The next signal that comes from the systems controller is either the test signal or the load subgrids signal. If the test signal is received, an on board hardware diagnostic test is conducted. When the algorithm component processor finishes with this test, it awaits the arrival of the count enable signal. When the count enable arrives, one of two different signals can be sent to the systems controller. The first of these signals is the test acknowledgement signal which signifies that a successful test has occurred and the count enable line is being passed on to the next algorithm component processor in the array. The other signal is the error signal. When this line is activated it notifies the controller that a hardware fault exists and that nothing can proceed until the fault is fixed.

The load subgrids signal can be activated without the testing of the algorithm component processors. If this mode of operation is followed, a hardware fault can go undetected and spurious results can occur. The applications program may want to do this though if the system has been running for awhile and a test has already been done. In any case, when the load subgrids signal is received by the algorithm component processor, a direct link between the controller and the processor's local RAM memory is established. The absolute location for the placement of the data in a memory write operation is directly controlled by the algorithm component processor.

The write cycle operation though is in the hands of the controller.

Each algorithm component processor stands ready to transfer subgrids but does nothing until the arrival of the count enable signal. The arrival of this signal automatically causes the algorithm component processor's wait line to be activated. This informs the controller that another algorithm component processor is preparing to receive its subgrid definitions. When the algorithm component processor is ready to receive its subgrids, it removes its wait line and waits for the arrival of write enable signals. The arrival of the write enable begins the synchronous loading (via write cycle) of the subgrid definitions. When the last subgrid has been loaded, the controller removes the write enable which notifies the algorithm component processor that it has received all of its subgrids. The algorithm component processor then sends the count enable signal high for the next algorithm component processor in the array.

When the last algorithm component processor in the array sends the count enable line high, the controller is formally notified that all algorithm component processors have been loaded and that the computation of contours can begin. The arrival of the compute contour signal is the next signal the algorithm component processor expects. Its arrival sets the operation of the algorithm component processor to the compute contours state.

During this state, each algorithm component processor computes contours from the subgrid definitions loaded in its local RAM. For the inputs and outputs during the compute contours state, there are two interrupts that are serviced. The input interrupt (highest priority) is initiated by the arrival of the input count enable signal. The output interrupt is initiated by the arrival of the output count enable signal.

The input interrupt signifies to the algorithm component processor that the input of a contour level is required. The interrupt handler resident in the processor's ROM conducts the input of this contour level in precisely the same manner as the input of subgrids. It does not require the saving of registers upon its initiation. (Note: The input of a new contour level forces the processor to start over in the production of coordinates and drawing instructions. Therefore, the register values are of no consequence and need not be saved.) Before exiting the interrupt, new register

values are provided which cause the compute contours algorithm to start with the very first subgrid in its RAM. Before exiting the interrupt, the processor activates the input count enable signal for the next algorithm component processor in the array. The whole process is estimated to take less than 1 microsecond and thus is the higher priority of the two interrupts.

The output interrupt is significant to the algorithm component processor in that it allows the processor to empty its output buffer, containing the coordinates and drawing instructions, onto the output bus in a very efficient manner. This interrupt is initiated when the output count enable signal is received by the algorithm component processor. The beginning of this interrupt involves the saving of all register values, all of which are restored upon exiting the interrupt. Before initiating the transfer with the display processing unit, the algorithm component processor determines how many bytes in its output staging area must be transferred. This staging area is in RAM memory and has a specified location known to the processor. The beginning address of this staging area can subtracted from the end address of the last known byte produced by the contouring algorithm. The difference obtained from this operation determines exactly how many bytes of information are to be transferred. Once the size of the output has been determined, the transfer process can begin.

The output operation to the display processing unit is done with a two line handshake. The algorithm component processor sets its ge.req line high which notifies the display processing unit it is ready to transmit data. If the display processing unit is ready to receive, it replies with the ge ack and a transfer of all the output for that particular algorithm component processor proceeds. (Note: The transfer sequence is covered in more detail when an explanation of the Silicon Graphics, Inc. Private Bus is given below.) When the transfer is completed, registers are restored, the output count enable is activated for the next algorithm component processor and a resumption of computations begins.

4.2.2. Algorithm Component Processor's Hardware

The original architecture of the algorithm component processor is fairly accurate but needs some modification now that its control lines have been more clearly defined. The new design

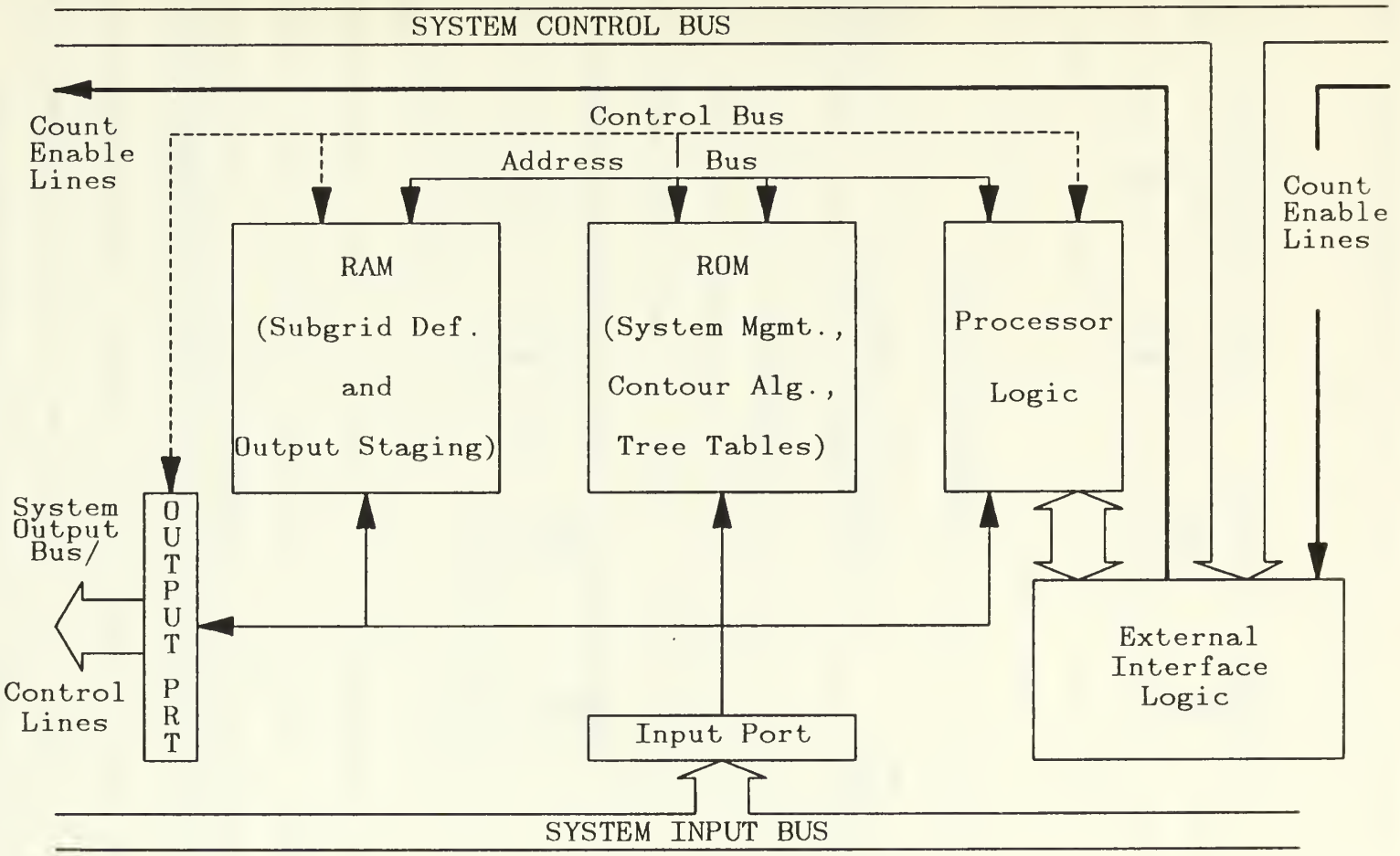

Figure 15.
Algorithm Component Processor for the Contour Surface Display Generator.

includes all the components of the original design with the addition of an external interface (Figure 15). The external interface logic determines the state of the algorithm component processor and manages the external signals for the chip resident processor.

The external interface logic determines the state of the algorithm component processor by supplying the address in ROM which contains the program for the indicated state of operation. The logic of the external interface identifies the state by making the appropriate hardware connections when one of the four state control lines is activated. If the reset line is activated, it provides a direct line to the processor indicating a reset operation. If the test line is activated, it provides several different lines for the processor (Figure 16).

When the test line is active, the external interface supplies the processor with the address necessary to run the test program. The external interface also makes physical connections available for the processor to communicate directly with the controller. The direct link between the algorithm component processor and the controller is not established, however, until the count enable line arrives at the external interface. When the count enable arrives, those physical connections are made and communication occurs between the controller and the algorithm component processor. When the count enable line is forwarded to the next algorithm component processor, the physical connections are severed.

In Figure 17, we see that the same kinds of connections are made for the load subgrids state. The signals which are exchanged between the controller and the algorithm component processor are different, however. The address of the ROM program which handles the input of subgrids is supplied to the processor when the load subgrids line is activated. The communication lines between the controller and the algorithm component processor are then established. When the count enable line arrives at the external interface, the physical connections are made and the transfer of subgrids takes place. When the count enable line is removed, these physical connections are severed and the processor goes idle waiting for the activation of the next state.

In Figure 18, we see that the final state involves a little more complexity. When the compute contours line is activated, all the lines necessary to configure the algorithm component

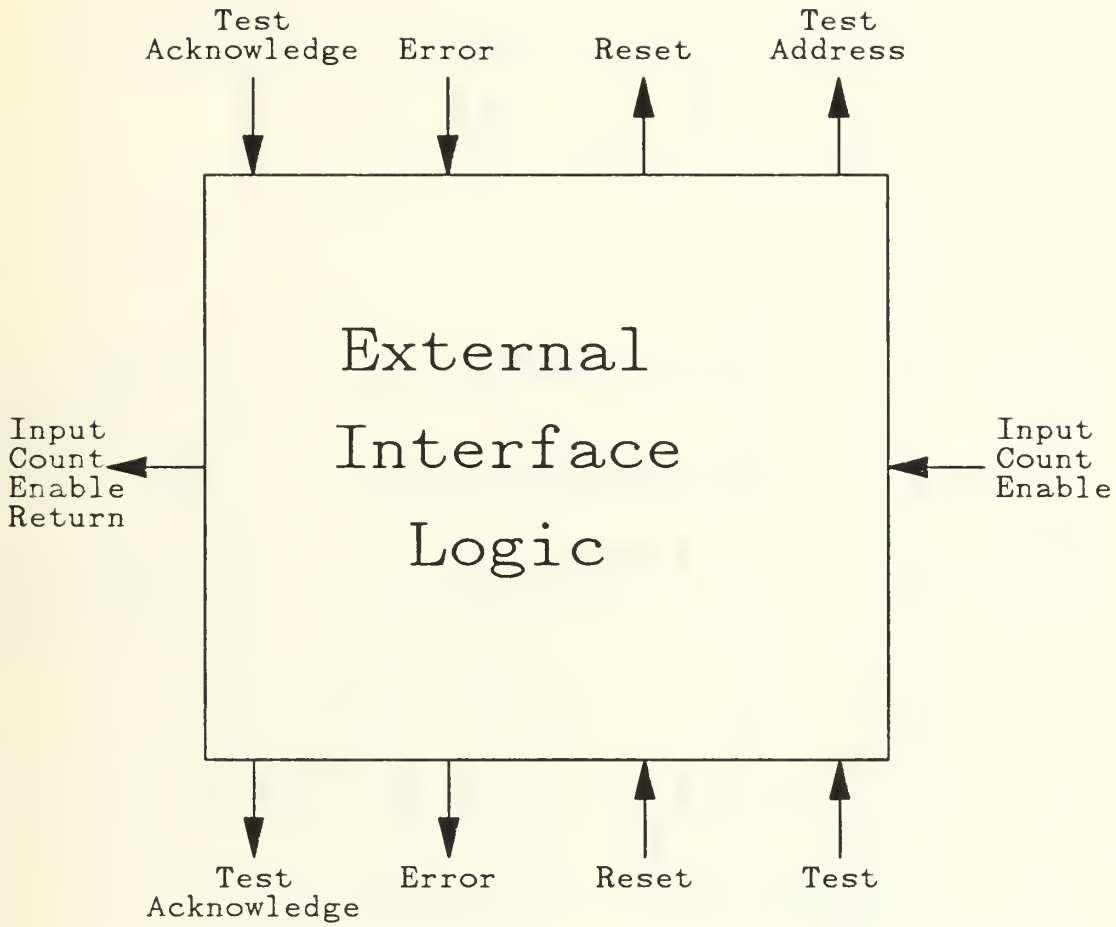

Figure 16

External Interface Logic for the Test State of the Contour Surface Display Generator

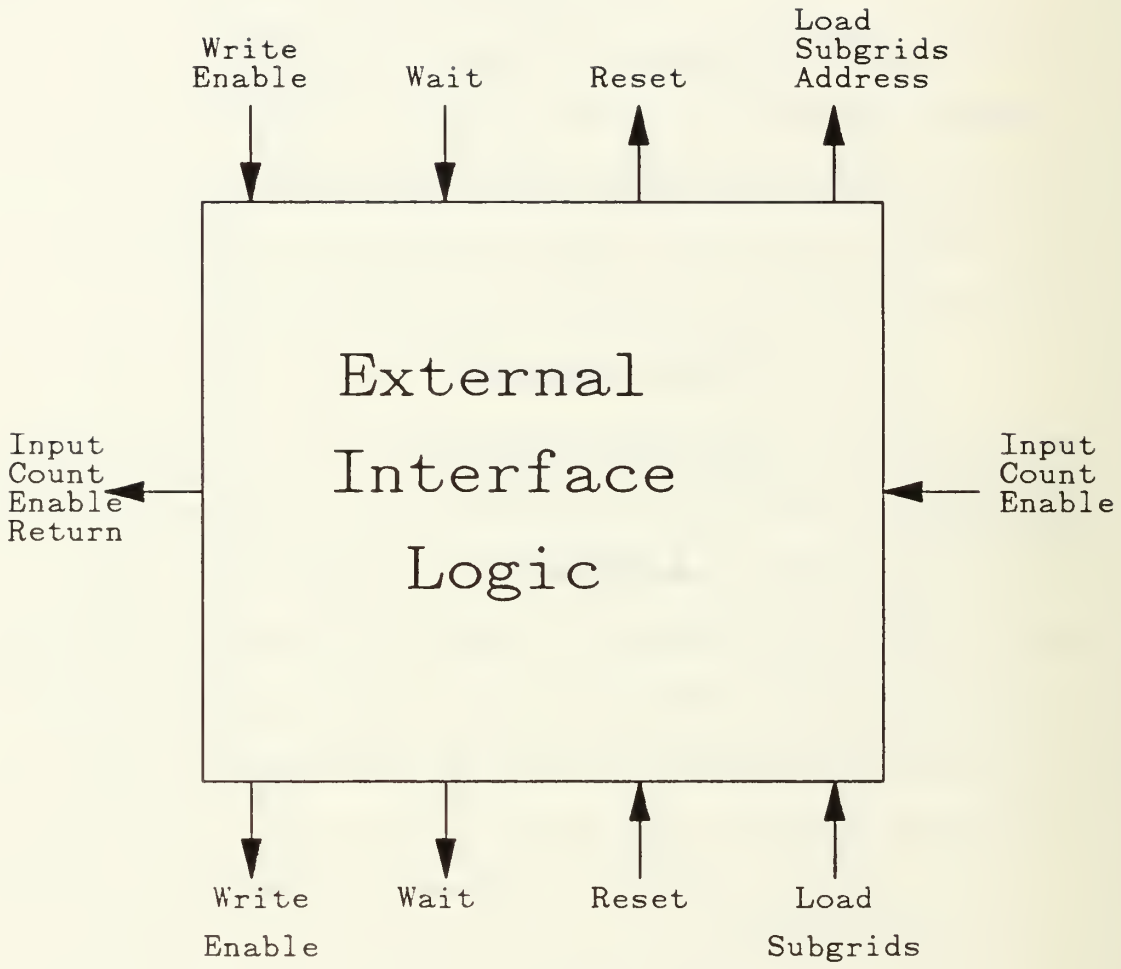

Figure 17

External Interface Logic for the Load Subgrids State of the Contour Surface Display Generator

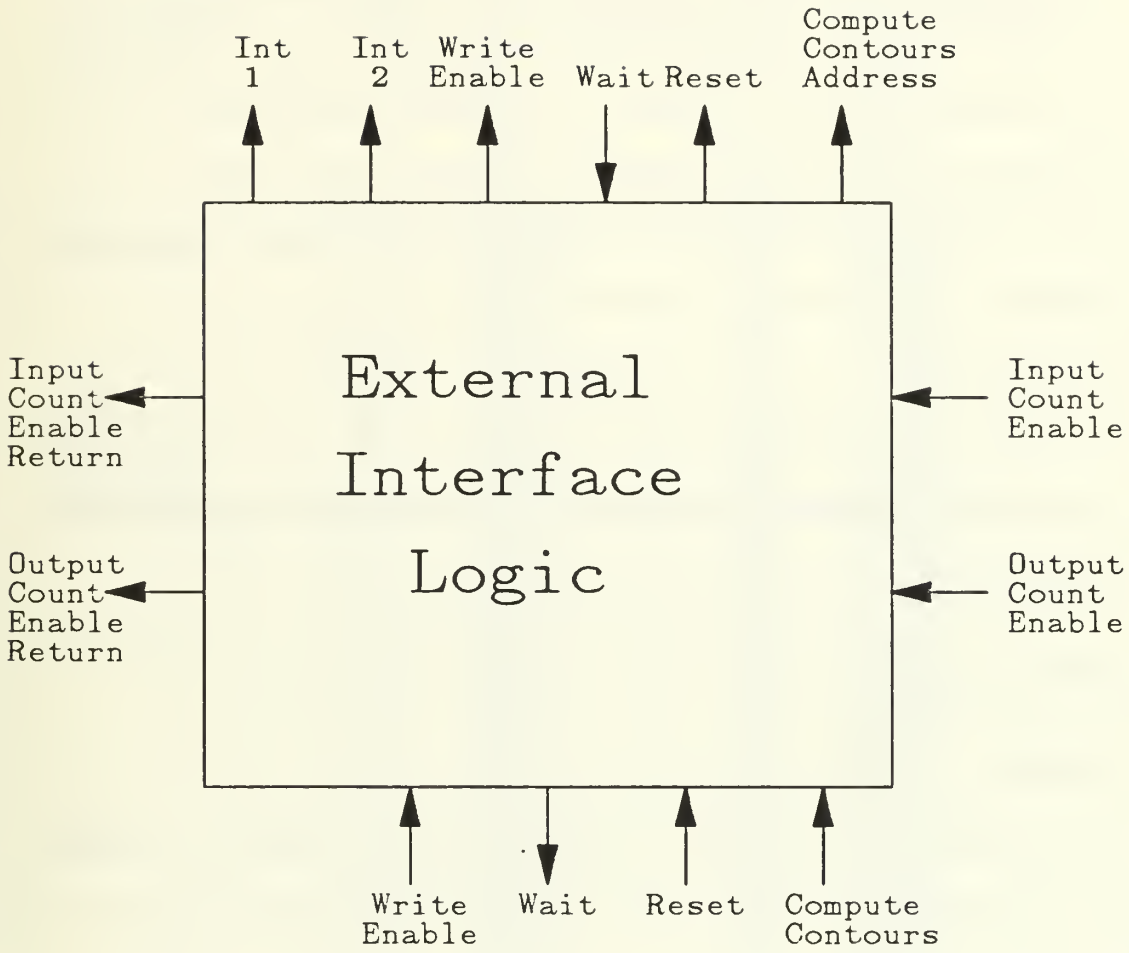

Figure 18

External Interface Logic for the Compute Contours State of the Contour Surface Display Generator.

processor to the compute contours state are connected. When either one of the count enable lines arrives (input or output), the physical connections are made, and interrupts are issued to the processor. The interrupt 1 line signifies a load contour level configuration and the interrupt 2 line signifies a dump coordinates and drawing instructions configuration. The interrupt 1 logic involves communication between the processor and the controller which requires the use of the lines established in the external interface logic. The interrupt 2, however, requires communication between the processor and the display processing unit. The lines needed to accomplish this are an integral part of the local control bus and are connected directly to the external port to the Private Bus (Figure 15).

All of these different configurations have one thing in common. The reset line is always present and when activated, zeroes all the registers and returns the algorithm component processor to the initial state.

4.3. System's Interfaces

4.3.1. Multibus

The contour surface display generator is connected to the Silicon Graphics, Inc. IRIS graphics system by means of the IEEE standard Multibus Backplane Bus (see Figure 19 and [15]). The Multibus interfaces to basically two different classifications of bus modules: (1) Masters - those modules which generate commands, and (2) Slaves - those which respond to commands. The parent processor (MC68000) is the Master module for the graphics system. The contour surface display generator is a slave module. A discussion of the slave module is necessary to understand how the contour surface display generator is integrated into the IRIS graphics system.

Slave modules respond to bus commands and are not capable of controlling the Multibus interface. The interface for the contour surface display generator functions as a slave module and can be divided into four basic elements: (1) Control (2) Address (3) Data and (4) Interrupts. By examining each of these elements with respect to their functional relationship to the Multibus interface, the mechanism the bus master uses to control the contour surface display generator can

	COMPONENT SIDE		CIRCUIT SIDE	
PIN Group	PIN	Description	PIN	Description
Power Supplies	1	Gnd	2	Gnd
	3	+5V	4	+5V
	5	+5V	6	+5V
	7	+12V	8	+12V
	9	-5V	10	-5V
	11	Gnd	12	Gnd
Bus Controls	13	BCLK/	14	INIT/
	15	BPRN/	16	BPRO/
	17	BUSY/	18	BREQ/
	19	MRDC/	20	MWTC/
	21	IORC/	22	IOWC/
	23	XACK/	24	INH1/
Bus Controls and Extended Address	25	RESERVED	26	INH2/
	27	BHEN/	28	AD10/
	29	CBRQ/	30	AD11/
	31	CCLK/	32	AD12/
	33	INTA/	34	AD13/
Interrupts	35	INT6/	36	INT7/
	37	INT4/	38	INT5/
	39	INT2/	40	INT3/
	41	INTO/	42	INT1/
Address	43	ADRE/	44	ADRF/
	45	ADRC/	46	ADRD/
	47	ADRA/	48	ADRB/
	49	ADR8/	50	ADR9/
	51	ADR6	52	ADR7/
	53	ADR4/	54	ADR5/
	55	ADR2/	56	ADR3/
	57	ADRO/	58	ADR1/
Data	59	DATE/	60	DATF/
	61	DATC/	62	DATD/
	63	DATA/	64	DATB/
	65	DAT8/	66	DAT9/
	67	DAT6/	68	DAT7/
	69	DAT4/	70	DAT5/
	71	DAT2/	72	DAT3/
	73	DATO/	74	DAT1/
Power Supplies	75	Gnd	76	Gnd
	77	RESERVED	78	RESERVED
	79	-12V	80	-12V
	81	+5V	82	+5V
	83	+5V	84	+5V
	85	Gnd	86	Gnd

Figure 19.

The Multibus Signals and Pin Assignments.

be best understood. Figure 19 shows the physical pin locations for the connection of the boards to the Multibus. These physical connections are standard for all boards on the Multibus backplane.

The control logic of the Multibus consists of circuits that forward the I/O and memory read/write commands to their respective destinations. This control logic also provides the bus with transfer acknowledge responses. The READ/WRITE commands are MRDC/, MWTC/, IORC/, and IOWC/ and are implemented as high speed buffers in the contour surface display generator [15]. The transfer acknowledge signal (XACK/) provides the bus master with a transfer acknowledge response once data provided by the bus master has been written. This same signal is provided when the read data requested by the bus master is available on the bus. This signal has a three state driver which is enabled by the board enable signal (BDEN/). The only other control signal for the slave module is the CCLK/ which is provided by the bus master. This signal generates the internal timing necessary to coordinate the transmit and acknowledgement of signals.

The address decoding logic decodes the appropriate Multibus address bits into RAM requests, ROM requests, or I/O requests. The contour surface display generator functions as both a RAM memory device and an I/O slave depending upon what actions are required by the bus master. The bus master therefore must have addresses available to it to access the ROM, RAM and I/O controllers. The selection of these addresses is determined when the system is pieced together and is not covered here.

The contour surface display generator both uses and produces data. Therefore, bidirectional drivers for data are required. These drivers are enabled by the I/O Read Command, I/O Write Command, Memory Read Command, and Memory Write Command.

The interrupt logic for the contour surface display generator is not very complicated because the only interrupt which affects it is the error interrupt given when the system fails. A simple non-bus vectored interrupt implementation is applicable [15].

4.3.2. Silicon Graphics, Inc. Private Bus

The Silicon Graphics Private Bus is a unidirectional, 16 bit bus intended to be totally dedi-

cated to the provision of coordinate and drawing instructions to the high speed Geometry Engines. The processors which produce the data for the Geometry Engines drive the Private Bus. Coordination of the transfer of data between these processors and the Geometry Engines is done via a two line handshake protocol.

The transfer sequence of the Private Bus involves the ge.req line which originates from the processor and the ge.ack which originates from the display processing unit. When the processor is ready to send its data, it activates its ge.req line. If the display processing unit can accommodate a transfer of data, it immediately returns the ge.ack. This return signal automatically commences the transfer process. If at any time during the transfer of this data either line becomes inactive, the transfer process is suspended. If the processor initiates this suspension of the transfer, it indicates to the display processing unit that it has no more data to send. The display processing unit responds to this by removing his ge.ack, thus ending the transfer process. If the display processing unit initiates this suspension, then it indicates to the processor that there is a problem with receiving more data. The processor then waits until the display processing unit is able to continue the transfer. The transfer process continues when the ge.ack is again activated by the display processing unit.

5. System Implementation

5.1. Overview of the IRIS Graphics System

In section 3. the target display system was stated as being the Silicon Graphics, Inc. IRIS. The IRIS's architecture was shown in Figure 6. The functions of the elements of that figure were not completely explained. Before we describe how the contour surface display generator is inserted into that system, we need to discuss the IRIS (Figure 20). (Note: This explanation is an overview. A more technical discussion is available in Silicon Graphics, Inc. technical publications.)

The IRIS graphics workstation is comprised of the UNIX operating system, the Ethernet communications network and a real-time three-dimensional color-raster graphics system. The

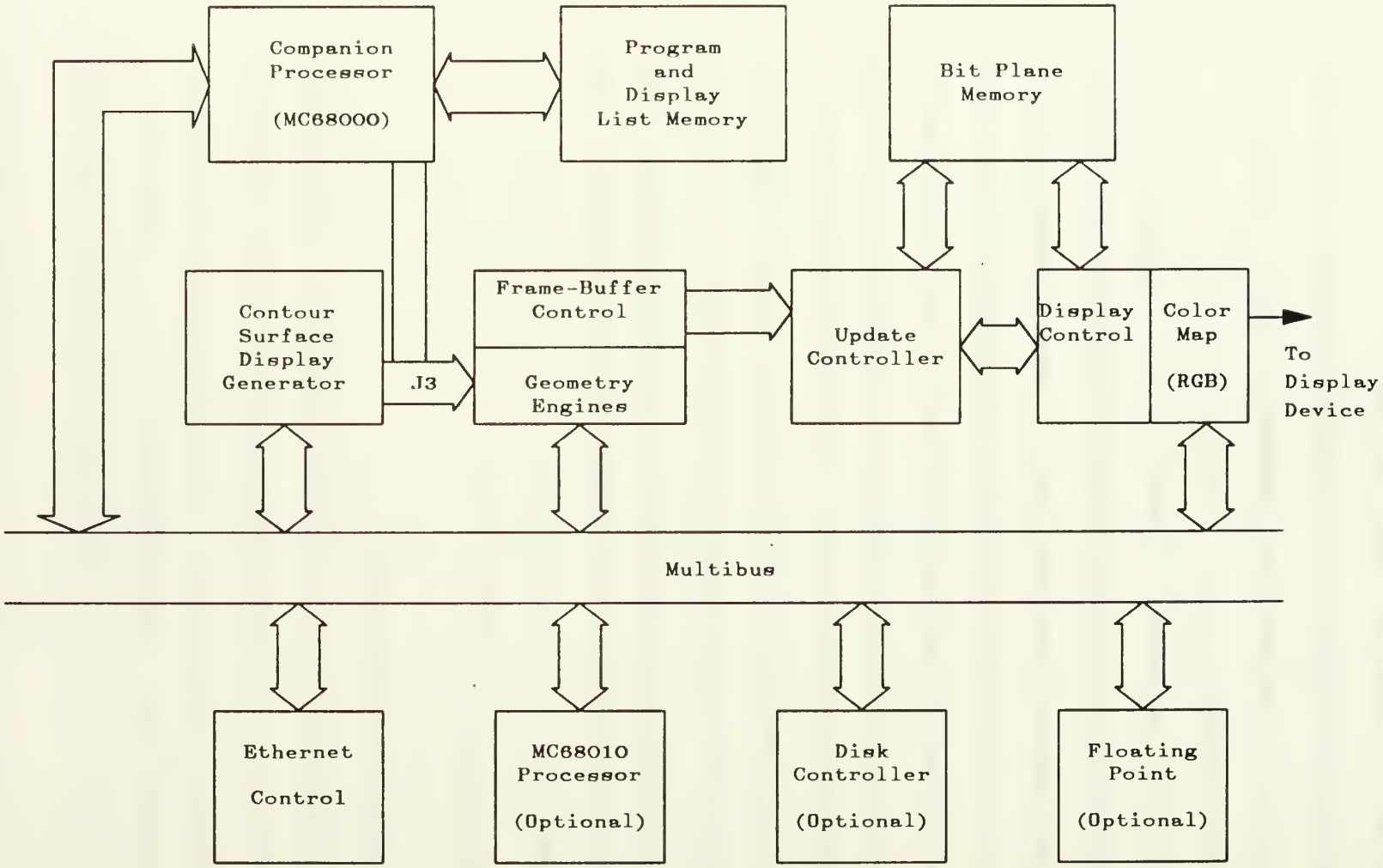

Figure 20
Contour Surface Display Generator Integrated into the Silicon Graphics, Inc. IRIS

hardware elements that make up the workstation are connected to one another via the Multibus. Graphics commands are issued by a host terminal on the workstation. The terminal uses the Motorola MC68000 as a controller and a Geometry Engine pipeline for matrix operations whose output is destined for a high-resolution color raster-scan display. The communications between devices is done on the Multibus. Graphics pipeline data is transferred on a private data bus [17].

Graphical output is initiated by the CPU by sending commands and data to the graphics pipeline. The Geometry Engines perform matrix transformations, clipping and scaling. The Frame Buffer controller interprets characters, controls fonts, and constructs lines and polygons. The update controller does scan conversion of polygons, lines and characters. The results of those operations are placed into the frame buffer. The display controller fetches the picture-element values from the frame buffer and draws them on the face of the color monitor [16].

5.2. Integration of the Contour Surface Display Generator

The Multibus Backplane supplies the power and inter-board communications capabilities required to implement an integrated graphics system. The contour surface display generator is constructed as a peripheral board and is added to the IRIS graphics system as a slave I/O and memory device. Figure 20 shows the contour surface display generator as an integral part of the IRIS graphics system.

The use of the contour surface display generator in the graphics system involves the establishment of a physical connection between the peripheral board containing the algorithm component processors and the display system. This connection involves the Private Bus port which transports the data directly to the Geometry Engines. In its present configuration, the IRIS system has a connecting cable that directly connects the system processor to the Geometry Engines (J3 connection of Figure 21).

When the contour surface display generator is added to the system, this physical connection must be shared by both itself and the system processor. To enable the user to alternatively route processor and generator data to the Geometry Engines, a hardware switch is added to the system.

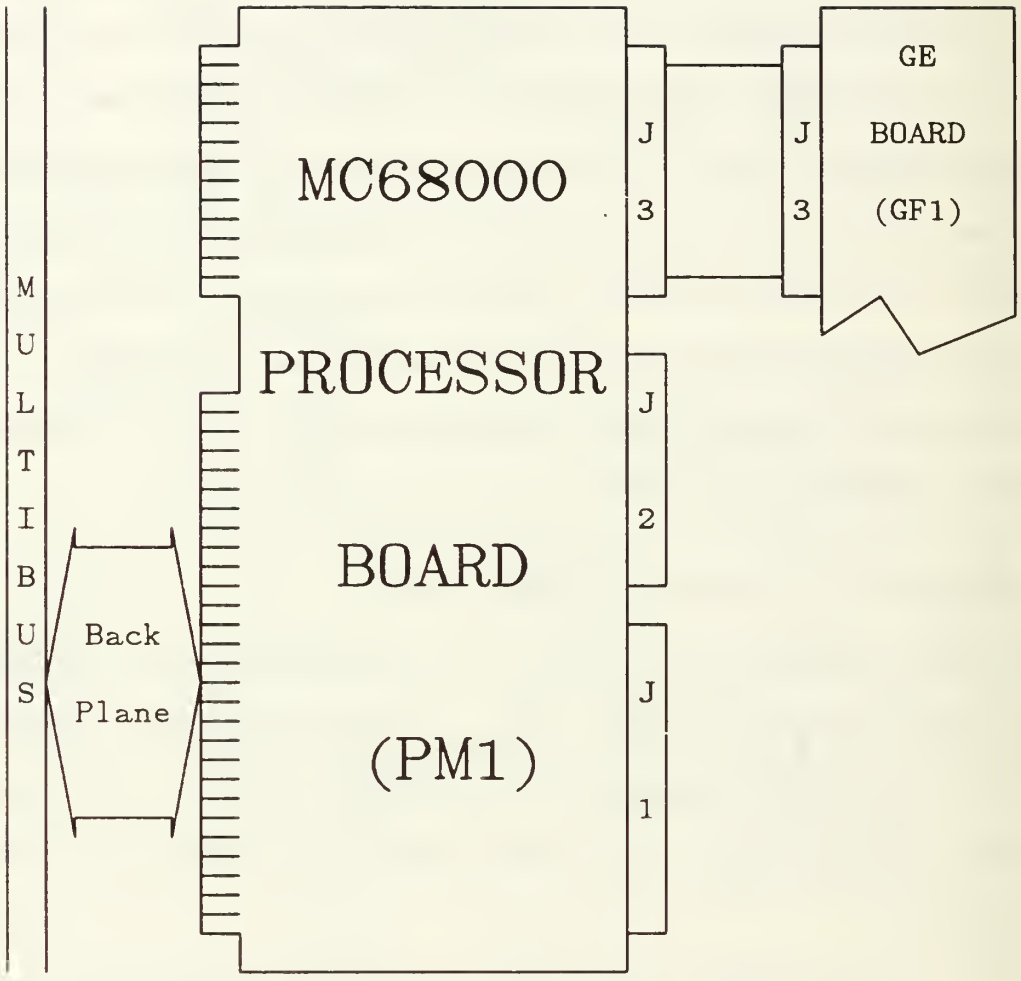

Figure 21

Silicon Graphics, Inc. IRIS Pipeline Connection for the Private Bus. (Courtesy of Silicon Graphics, Inc.)

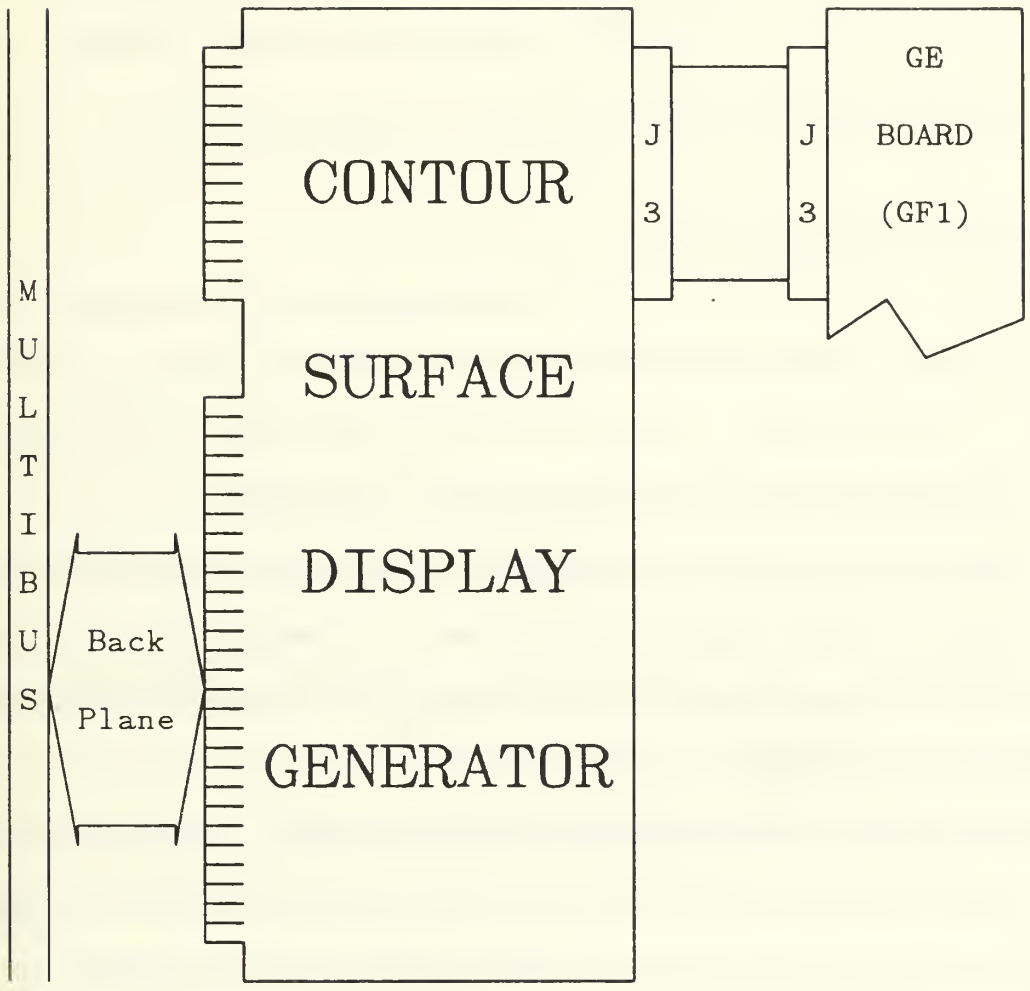

Figure 22

The J3 Pipeline Connection of the Silicon Graphics, Inc. Private Bus for the Contour Surface Display Generator.

This hardware provides the system with a way to multiplex the direct path of the Private Bus. A software switch then provides the control of the Private Bus' origin and configuration. When activated, this switch establishes a path from the contour surface display generator (J3 connection of Figure 22). If it is not activated, the IRIS system remains in its original configuration.

5.3. Determining the Total Number of Algorithm Component Processors

5.3.1. Review of Known Parameters

In order to test the feasibility of constructing the contour surface display generator on a single board, we need to determine the number of components required, i.e. algorithm component processors. We accomplish this by reviewing the parameters obtained from the original study and relating those parameters to the technical assumptions made in this study [1].

The first of these parameters is the number of 2 x 2 subgrids in a 30 x 30 x 30 three-dimensional grid. The total number is determined by multiplying the total number of two-dimensional cross-sections (90) times the total number of 2 x 2 subgrids per cross-section (29 x 29). This total is 75,690 2 x 2 subgrids.

The next parameter that is necessary is the maximum number of these subgrids that are executed for each change in contour level. The maximum observed percentage of 2 x 2 subgrids that generate coordinates is 13 percent or approximately 9900 of the total 75,690 subgrids.

The next parameter we need is the average number of coordinates and drawing instructions that the set of 9900 subgrids generates. The three-dimensional coordinate plus drawing instruction is termed a quadruple. The average number of quadruples generated for the set of 2 x 2 subgrids that generate coordinates is 2.54 quadruples. Multiplying this number by 9900, we find that the average contour surface display contains 25,146 quadruples.

Another parameter we need is the maximum number of memory references required to compute one execution of the contour surface display generation algorithm. The number of memory references is 2650 [1]. To calculate the computation time of the algorithm from this value, a memory access time needs to be specified. The access time specified was 250ns [1]. Since that

time, systems have been produced with access times as small as 50ns [18]. By assuming that the algorithm component processor uses a 50ns static RAM to store its subgrid definitions, we can set the maximum computation time for one 2 x 2 subgrid at 132.5 microseconds.

The final parameters that need to be addressed involve the input and output of the contour surface display generator. Both the input and output buses are designated as 16 bit data paths. The only input that affects the real-time capabilities of the system is the input of one 32-bit contour level for each execution cycle. (Note: An execution cycle consists of the computation of the maximum number of subgrids that generate coordinates and drawing instructions.) The input of this contour level is not a major factor in the calculation of the number of algorithm component processors required.

The output, on the other hand, is highly dependent upon the number of subgrids executed. Each execution of the contouring algorithm on a subgrid generates between zero and six coordinate and drawing instruction quadruples. The zero quadruples case only requires a boundary check on the subgrid. The other cases, however, involve computation and output. Each quadruple output is 13 bytes in length. As detailed above, the average size of the output from a single 2 x 2 subgrid is 2.54 quadruples. The average output size is therefore approximately 34 bytes and the total number of output bytes for one execution cycle is 336,600 bytes.

The output size does not greatly affect the number of algorithm component processors, but it does affect the minimum clocking rate of the output bus. (Note: The time required to transfer output is relatively small when compared to the computation time required to execute the contouring algorithm.) For a data path of 16 bits, 168,000 transfers are required to send 336,000 bytes of data to the display processing unit. The output bus must be clocked at a rate no less than 5Mhz. In fact, the bus has to be clocked at a rate much higher than this to accommodate the handshaking time required between the algorithm component processor and the display processing unit. For calculating output times, a 10 Mhz clock time is assumed.

5.3.2. Determining a Realistic Number of Processors

In determining how many processors the contour surface display generator needs, the parameters developed above must be taken into consideration. In addition to this, we use two terms to describe the process of determining the optimum number of processors needed. The first term, throughput rate (TR), is a common term used in the calculation of a computer's efficiency. The second term, partial throughput rate (PTR), is a term that requires some additional explanation.

The throughput rate of the contour surface display generator pertains to one algorithm component processor and is the sum of three different computational parts. The first of these is the total time required to input one contour level. The second part involves the time it takes for the optimal number of 2 x 2 subgrids to be computed in serial and remain under the real-time threshold value of 33.333 microseconds or one-thirtieth of a second. The last of these parts involves the amount of time required to output the coordinates and drawing instructions produced by the computations. The time to input a contour level is insignificant in the computation of the throughput rate. The combination of the computation time and the output time, however, does have a direct bearing on the real-time capabilities of the system.

To find this optimal number of subgrids, we need to apply another term. The partial throughput rate of the contour surface display generator consists of the time it takes to complete one computation of the contouring algorithm and output the corresponding coordinates and drawing instructions. By combining these two factors, a serial computation time can be derived which yields the optimal number of subgrids. The computation time comes directly from the parameters specified above. The output time is derived by adding together the handshake protocol time for each output and the 34 bytes of coordinates and drawing instructions that are to be transferred. The total is then applied to a 10 Mhz clock rate which gives us an output time of 2.1 microseconds.

PTR = Computation Time + Output Time

= 132.5 microseconds + 2.1 microseconds

= 134.6 microseconds

Since we know that the upper bound of the throughput rate (33.333 milliseconds) and the partial throughput rate (134.6 microseconds), the number of serial computations that can be accomplished by one algorithm component processor is found by simple division, 240. These 240 computations represent the 13 percent of total subgrids loaded in the algorithm component processor that generate output. From this, we obtain the total subgrids which can be allowed in one algorithm component processor (1846). By dividing the total number of subgrids (75,690), we determine that the contour surface display generator only requires 41 algorithm component processors. This number of processors is much more manageable than the 1000 processors specified in [1].

5.3.3. Other Factors Affecting System Performance

Other factors exist which affect system performance in addition to the throughput rate. Of the 1846 subgrids loaded into an algorithm component processor, 87 percent do not produce any output. Each time a contour level is loaded, these subgrids must be evaluated by the processor to determine whether or not they are producers of output. This evaluation requires time and adds some overhead to the system throughput rate.

If we assume that each of these evaluations requires 5 memory references, then to evaluate 87 percent of the subgrids takes an additional 401.5 microseconds. When we add this in to the calculations of the throughput rate, we see little impact on the total number of processors in the contour surface display generator.

Another factor which affects the number of algorithm component processors in the contour surface display generator is the distribution of the subgrids in the processors. All of the parameters up to this point have been based on a very simple assumption. This assumption is that the subgrids that generate output are uniformly distributed among the algorithm component

processors.

This uniform distribution allows a statistical approach in the determination of the optimal number of algorithm component processors. To be realistic, this uniform distribution of subgrids is virtually impossible. The gives rise to the question, how can we distribute the subgrids to obtain a moderate level of uniformity?

For the distribution of the subgrids, we adopt an ad hoc method rather than a precise method because of the variability of factors which influence this distribution. First of all, the size of the contour surface display changes with the application. The proportionality of that object in three-space also varies with application. These factors influence the amount of non-output producing subgrids that remain in the cubic dimensions of the subgrid definitions.

The ad hoc method recommended is to begin by taking one 30 x 30 slice from the end of the three-dimensional grid (see Figure 23). Chances are that that slice will be a non-output producing slice. The next slice to take comes from the center of the three-dimensional grid. Subgrids on this slice are more likely to contain a large number of contours. The next slice to take is the slice inside of the previously loaded end slice. The next slice to be loaded is the one outside of the middle slice taken, etc.. As the loading process continues, the slices which originated on the end approach the center as the slices which originated in the center approach the other end. This loading process is done a total of three times, once for each set of two-dimensional cross-sections.

This distribution method is not fail proof and does not guarantee the uniform distribution required. If any one of the algorithm component processors has to execute more than the allotted maximum number of computations (240 with 41 processors), the real-time capability of the contour surface display generator is compromised by 134.6 microseconds for every extra computation over the allotted 240.

To allow for this type of variance, another 9 algorithm component processors are added to the system. This brings the total number of algorithm component processors up to 50 and fixes the total subgrid count at 1514 per processor. The maximum number of executable subgrids, (i.e. 13 percent of 1514), is then 197. This provides a computational tolerance of 43 subgrids from the

previously specified maximum for which the system has time (240).

5.4. Revised Circuit Complexity Estimate

From previous research, we obtained a circuit complexity estimate in which only 8K of Dynamic RAM, 8K of ROM and processor logic were required to implement the algorithm component processor [7]. In this study, several elements have been added to the system. In addition to the circuitry proposed, we added: 24K more dynamic RAM to accommodate the additional space for the 1514 subgrids, 4K of microcoded ROM to manage the states of the algorithm component processor, a resident test bed to do self diagnostics, an external interface logic unit to control external communications and some additional space for refresh logic for the dynamic RAM of the algorithm component processor. All of this requires that we reassess the algorithm component processor's circuit complexity.

The total number of devices on a VLSI chip is dependent upon some key assumptions. The first assumption is that static RAM implementation is not feasible at this point because of the vast amount of space it requires on a single chip. We, therefore, assume that dynamic RAM is the most feasible for our purposes because it only requires two devices per bit versus the four or five devices per bit of the static RAM. This does add another problem however. We now must include, as part of the processor logic, some circuitry to manage the refreshing of the dynamic memory cells. We assume that this adds at least 5,000 devices to the 18,000 devices previously specified [10].

In addition to the 23,000 devices needed to implement the on board processor, we have added some additional memory and interface circuitry. (Note: Figure 24 illustrates the device count of the algorithm component processor and the other associated device counts.) We added 24K bytes more RAM and 4K bytes more ROM which brings our total memory requirement to approximately 623K devices. Adding in the additional device count for the interface logic and test facility our device total is now in excess of 660K devices. This number is still well below the 2 million devices per chip level that is being produced in research laboratories [13]. The design of this system is, therefore, well within the grasp of current technology.

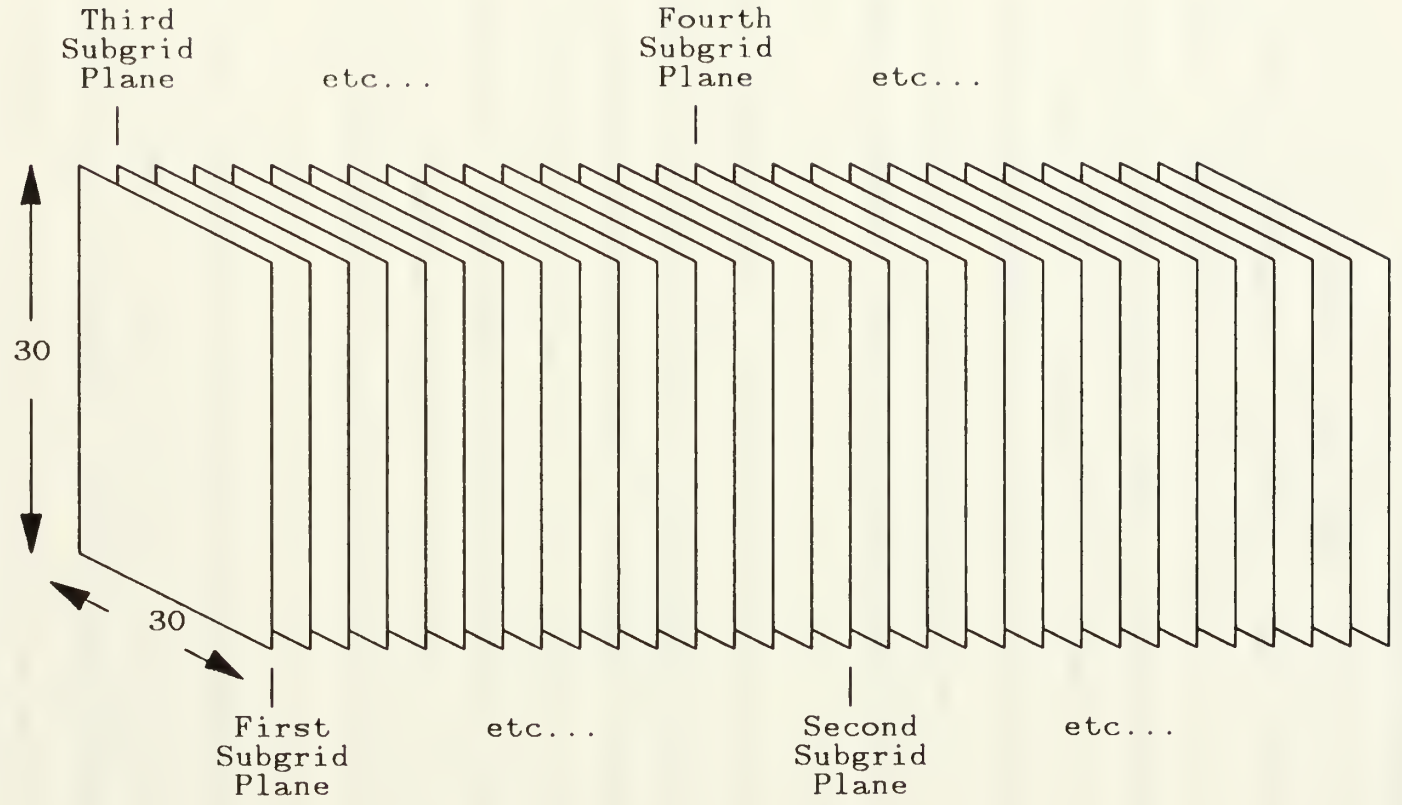

Figure 23
A Loading Methodology for the Uniform Distribution of Subgrids in the Contour Surface Display Generator (Y-Z Planes Shown).

(1) RAM space -- (2 devices/bit)		
8192 x 32 bits	=	524,288 devices
(2) ROM space -- (1 device/bit)		
-- Tree tables		
2048 x 16 bits	=	32,768 devices
-- Microcode		
2048 x 32 bits	=	65,768 devices
(3) Processor space --		
-- ALU		
-- Register block		
-- Control section		
-- Data, address and control buses		
-- Refresh logic		
	=	23,000 devices
(4) Interface Unit and Test Bed --		
-- External Interface Logic		
-- Test circuitry		
-- Latches and Drivers		
	=	15,000 devices
Device Total	=	660,582 devices

Figure 24
Algorithm Component Processor's Circuit Complexity Estimate

6. Further Research Topics

6.1. A Fault Tolerant System

In the design of the contour surface display generator, little was done to allow for system recovery in the event of an error. We dedicated only one control line, the error control, which suspends operation of the entire system if one algorithm component processor has an error condition. This configuration is totally unacceptable unless we can guarantee that the algorithm component processors are highly reliable. An option that we have is to include in our design the ability to recover from any error and allow only minor degradation to the system.

The introduction of fault tolerance, or error recovery, into the design of the contour surface display generator is a complex issue. There are many tradeoffs that must be considered in implementing a fault tolerant system. The most revealing of these tradeoffs is the economic impact of implementing or not implementing error recovery.

If error recovery is implemented, it adds a great deal of complexity to the circuitry of both the controller and the algorithm component processor. It also requires that additional algorithm component processors be added to the system for redundancy purposes. This redundancy, in turn, is what gives us a fault tolerant system. The precise implementation of the fault tolerant system is beyond the scope of this study but deserves more study. Some of the issues surrounding its design, however, are included below.

6.1.1. A Fault Tolerant Contour Surface Display Generator

The initial design of the contour surface display generator included the integration of a test module in each of the algorithm component processors. These test modules are intended to detect faults in the system before any computing is started. These mechanisms are for fault avoidance and not fault tolerance.

The introduction of fault tolerance involves the addition of control lines to handle any occurrence of an error at any time after the initial test. These lines are activated when an error condition is detected. The detection of an error is accomplished by an on board test module.

This burdens the hardware designer with the additional complexity of the design of an effective test module. This is the first part in the design of a fault tolerant system, i.e. how to detect an error.

Once the error detection problem is solved, the next step is to integrate the additional error recovery lines mentioned above into the contour surface display generator. With these new control lines, the controller manages the disconnection of the faulty algorithm component processor, the connection of a redundant algorithm component processor, the reassignment of subgrids and the resumption of operations.

The last step in the implementation of an error recovery system is the device level implementation. This involves the addition of circuitry in the algorithm component processors which suspends operations until all error conditions are cleared. This circuitry also enables the controller to remotely disconnect the count enable lines to the algorithm component processor. (Note: Once the count enable lines are disconnected from an algorithm component processor, that processor no longer has any way to input or output data.) The addition of circuitry to the controller is necessary to enable it to control the error correction process.

6.1.2. A Summary of Design with Error Recovery

The inclusion of error recovery capability adds a great deal of complexity to the design of the contour surface display generator. This complexity can be attributed to the following areas:

(1) The design of an effective test module which not only includes error avoidance but error detection during the system's operation.

(2) The addition of control lines to administrate the error recovery process. This requires additional space on the peripheral board to accommodate the extra lines.

(3) The increased complexity of the circuitry external to the controller which is needed to physically connect the error recovery control lines to the algorithm component processors.

(4) The modification of the external interface logic of the algorithm component processor to accommodate the connection or disconnection of the count enable lines from the controller.

(5) The addition of the ability for the controller to replace the subgrids of a deficient algorithm component processor and place those subgrids into one of the redundant algorithm component processors. This involves memory to memory transfer between the failed processor and the controller or a direct transfer of the subgrids from the failed processor to the good processor.

(6) The addition of the redundant algorithm component processors to the system complicates the process of loading the subgrids. (The redundant algorithm component processors must be explicitly disconnected when not in use by disabling their count enable lines.)

Although these complexities compound the design problem, the resolution of their implementation makes the contour surface display generator a much more reliable system.

6.2. A More Efficient System

6.2.1. Technology Dependent System

If we closely examine the arguments of the previous section, we see that a majority of time spent during the compute contours state involves the execution of the contouring algorithm. In fact, less than 2 percent of time is spent doing input and output while the remaining 98 percent is spent doing computations. The number of serial computations which can be accomplished in real-time is dependent strictly on two things: the number of memory references required for one execution of the contouring algorithm (covered in more detail below) and the amount of time required for one memory reference.

A 50 nanosecond access time was assumed. Because this is the state of the art for technology today, we cannot assume a lesser value. As the technology becomes available, the improvement of access times will be the driving factor in determining the number of algorithm component processors. The decrease in memory access times somewhat parallels the increase in device quantity available on a single microchip. This phenomenon is beneficial for the contour surface display generator because with this increase in device capability, more algorithm component processors can be produced on a single chip. We will need more memory on the chip when the access times decrease because more subgrids will be loaded into a single algorithm component processor.

We can see that the evaluation of the number of algorithm component processors needed is dynamic with respect to VLSI technology advances. The thing the designer should be concerned with now is to apply existing technology to the development of the portions of the contour surface

display generator which will remain static as the system evolves. The static portions include: the central processor, the microcoded ROM, external interface circuitry and error detection/correction circuitry.

6.2.2. The Contouring Algorithm

In section 5, we made a decision which directly affected the choice of how many algorithm component processors were needed. The assumption made was that every subgrid executed would use the maximum number of memory references (2650) in its execution. Furthermore, this maximum number assumes that each subgrid that executes produces the maximum number of coordinate and drawing instruction quadruples (6 quadruples). As a worst case argument, this assumption is valid. Practically, however, this is not the case.

We learned above that statistically only 13 percent of the subgrids in an algorithm component processor produce anything and that there is an average of 2.54 quadruples generated for each of those subgrids in that 13 percent. We have no way of knowing how many references are required to produce a single output quadruple and we cannot therefore determine the average number of references needed for a single execution of the contouring algorithm.

What needs to be studied is the dynamic operation of the contouring algorithm, and the flow points of that algorithm. By evaluating the flow points, we can determine an average number of memory references required, and thus, establish a more feasible figure to use in determining the number of algorithm component processors needed for the contour surface display generator.

6.3. Refinement of Design

This study is intended to act as a catalyst in the implementation of the contour surface display generator. Certain design decisions were made which directly affect how some aspects of the system are to be developed. Much work is still required before this system can be realized. Research includes both the hardware and software aspects of the system.

6.3.1. Hardware Topics

The hardware implementation of the contour surface display generator has been generally specified. There are three main elements which need refinement. The elements involve the formal VLSI design of the algorithm component processor, the integration of the Motorola MC68000 microprocessor and the MC68230 parallel interface/timer (PIT) to manage the contour surface display generator's operation, and the integration of all the components and support logic into a peripheral board compatible with the Multibus Backplane.

The design of the algorithm component processor involves the use of computer aided design (CAD) in order to formalize the circuitry of the chip. The design effort involves the breadboard design of the algorithm component processor's control logic, external interface logic, test bed and refresh logic. The integration of these elements with RAM and ROM memory in an orthogonally designed VLSI component is vital to the efficient operation of the contour surface display generator.

The control logic of the algorithm component processor should closely resemble the functionality of the Motorola MC68000 for ease of integration with the controller. The external interface logic should include all the elements discussed earlier in this study and it should include all aspects of off-chip communications (i.e. buffers, drivers and internal latches). The test bed's implementation is dependent upon the extent of fault tolerance desired for the component. As a minimum, it should be able to detect a hardware fault before leaving the test state.

The implementation of the refresh logic is critical to the efficient operation of the component's control logic (or processor). It must be integrated into the component in such a manner so as not to disrupt the processor's execution of the contouring algorithm. The refresh logic is therefore a difficult part of the implementation and adds a great deal of complexity to the design of the system.

The next component which needs further refinement is the systems controller. We have indicated that the controller consists of the MC68000 microprocessor and MC68230 PIT. Other circuitry must be included as an integral part of the controller. ROM memory is required to

contain the program which runs the contour surface display generator. RAM memory is required to stage subgrids locally before they are to be loaded directly into the algorithm component processors.

The other circuitry required in developing the controller involves the logic between the controller and the algorithm component processors. The implementation of this circuitry requires some tradeoff decisions. The circuitry determines how the control lines must be interfaced to the algorithm component processor. The key question is: Are they designed for simultaneous arrival or are they cascaded into each processor? We know that the lines which configure the states of the contour surface display generator must be configured for simultaneous arrival. The rest of the control lines and their implementation is contingent upon the evaluation of tradeoffs or functionality.

The arrangement of the above components and support devices on the peripheral board is of major concern. The questions to be answered are related to whether or not all of the components can fit on one board. The options available if they cannot fit on one board involve one of two alternatives. Either a multimodule hookup of some sort can be devised to keep the components in very close proximity to each other or another board must be added to the system to accommodate the overflow. The multimodule alternative is definitely the most promising of the two. An extra board should only be used if the overflow cannot be accommodated by the multimodule method. In any case, this issue deserves further study and is highly dependent on the final assessment of what circuitry is needed in the final configuration of the contour surface display generator.

6.3.2. Software Topics

Software for the contour surface display generator needs to be developed to control both the controller and the algorithm component processor. The controller's software is resident in the controller's local ROM. It contains the program necessary to interface with the Multibus and the algorithm component processors. It contains the appropriate sequencing of events for each state of the contour surface display generator's operation.

The above functions of the software for the controller are not all inclusive. They generally specify that the programming of the controller is necessary so that the system can be fully implemented. The specifics of this software development effort is beyond the scope of this study.

The software for the algorithm component processor can be divided into two parts. The first part involves the development of the managerial software which controls the sequencing of the states of the algorithm component processor. The second of the two parts is an extension of what was described earlier as the contouring algorithm. This algorithm must be studied in order to arrive at the average number of memory references per subgrid executed. The integration of this algorithm into the ROM of the algorithm component processor and the managerial software is trivial. This software development is beyond the scope of this study.

7. Conclusions

This study is intended to give a clearer system description of the elements comprising the contour surface display generator. Some of the deficiencies of earlier research were remedied by clearly defining the control lines of the system. The technological feasibility of the system's implementation was established in two ways. A realistic number of algorithm component processors was established (50). The circuit complexity of the algorithm component processor chip was established for those 50 processors that is within the capabilities of current VLSI technology.

In our study, a methodology was developed by which control lines were added according to their functional need. This functional need was established by dividing the operation of the algorithm component processor into states. A state diagram was presented to indicate the operational states of the hardware. The state diagram, once developed, was the foundation for establishing the data flow of the system. From this data flow description, control lines were established to implement the data flow of the system.

The establishment of the control lines and their functionality dictated that other hardware was needed. This hardware took the form of a systems controller. Once all the internal hardware of the system was defined, the target graphics system, the Silicon Graphics, Inc. IRIS, was

described. The interfaces that are particular to the IRIS System (Multibus and Private Bus) were firmly established with their applicable protocols. (Note: This bus arrangement fit nicely with our definition of separate input and output buses.)

Before we could realistically build the contour surface display generator, a smaller number of algorithm component processors had to be established. Implementation of 1000 processors in a Multibus configuration surely was not feasible. Statistical data from past execution of the contouring algorithm on a large uniprocessor system provided for a substantial reduction in the number of algorithm component processors required [4]. Using these statistics, 41 processors was established as the maximum number of processors needed. Upon further study, it was determined that this number may be insufficient because of the unpredictable nature of the input of subgrids to the system. The end result was to pad the number of processors by 9 to ensure the system's real-time capabilities were secure.

The enumeration of other factors that effect the design of the contour surface display generator was also included. Among these was the clear definition of the data flow behavior of the contouring algorithm. Another factor included the question of whether or not fault tolerance was to be considered in the development of a more reliable contour surface display generator.

The study answered many of the questions present in previous research. It also addressed a number of the issues involved when developing a peripheral of an integrated computer system. It was the intention of this study to act as a catalyst to spur research in the resolution of these issues. The desired end result is the successful development and implementation of a real-time contour surface display generator.

8. References

1. Zyda, Michael J. "The Feasibility of a Multiprocessor Architecture for Real-Time Contour Surface Display Generation," Technical Report NPS52-84-025, Monterey, California: Department of Computer Science, Naval Postgraduate School, December 1984.

2. Barry, C. D. and Sucher, J. H. "Interactive Real-Time Contouring of Density Maps," American Crystallographic Association Winter Meeting, Honolulu, March 1979, Poster Session.

3. Faber, D. H., Rutten-Keulemans, E.W.M., and Altona, C. "Computer Plotting of Contour Maps: An Improved Method," *Computers & Chemistry,* Vol. 3, pp. 51-55, Great Britain: Pergamon Press Ltd., 1979.

4. Wright, T. and Humbrecht, J. "ISOSRF -- An Algorithm for Plotting Iso-Valued Surfaces of a Function of Three Variables," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM,* Vol. 13, No. 2 (August 1979), pp. 182-189.

5. Zyda, Michael J. *Algorithm Directed Architectures for Real-Time Surface Display Generation,* D.Sc. Dissertation, Dept. of Computer Science, Washington University, St. Louis, Missouri, January 1984.

6. Zyda, Michael J. "A Decomposable Algorithm for Contour Surface Display Generation," Technical Report NPS52-84-011, Monterey, California: Department of Computer Science, Naval Postgraduate School, August 1984.

7. Zyda, Michael J. "Real-Time Contour Surface Display Generation." Technical Report NPS52-84-013, Monterey, California: Department of Computer Science, Naval Postgraduate School, September 1984.

8. Zyda, Michael J. "A Contour Display Generation Algorithm for VLSI Implementation," *Selected Reprints on VLSI Technologies and Computer Graphics,* Compiled by Henry Fuchs, p. 459. Silver Spring, Maryland: IEEE Computer Society Press, 1983.

9. Zyda, Michael J. "A Contour Display Generation Algorithm for VLSI Implementation," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM,* Vol. 16, No. 3 (July 1982), p. 135.

10. Zyda, Michael J. "Multiprocessor Considerations in the Design of a Real-Time Contour Display Generator." Technical Memorandum 42, St. Louis: Department of Computer Science, Washington University, December 1981.

11. Newman, William H., and Sproull, Robert F. *Principles of Interactive Graphics,* Second Edition. New York: McGraw-Hill, 1979.

12. Zyda, Michael J. "Joystick Driven Display Rotation and Control Console Management," Technical Memorandum 24, St. Louis: Department of Computer Science, Washington University, November 1980.

13. Micronews "IBM Experimental Million Bit Memory Chip." *IEEE Micro,* Vol. 4, No. 4 (August 1984), p. 119.

14. Uhr, Leonard *Algorithm-Structured Computer Arrays and Networks,* Orlando, Florida: Academic Press, 1984.

15. Intel Corporation Technical Publication. *Intel Multibus Specification,* Intel Corporation, Santa Clara, California, 1982.

16. Liebson, S. *The Handbook of Microcomputer Interfacing,* First Edition, Blue Ridge Summit, Pennsylvania: Tab Books Inc., 1983.

17. Clark, J. H. and Davis, T. "Workstation Unites Real-Time Graphics with Unix, Ethernet," *Electronics,* October 20, 1983.

18. Inmos Corporation Preliminary Data, *IMS T424 Transputer,* Inmos Corporation, Colorado Springs, August 1984.

19. MacGregor, D., Mothersole, D., and Moyer, B. "The Motorola MC68020," *IEEE Micro,* Vol. 4, No. 4 (August 1984), p. 101.

9. Appendix A

Algorithm for the Contour Surface Display Generator

Systems Controller

The following algorithm describes the functional behavior of the systems controller of the contour surface display generator.

{RESET STATE}

1. Power On

{Reset = high}

{TEST STATE}

2. If (Master = message) then

{'Test desired? yes/no'}

If (Test = yes) then

Test_line = high Counter = zero

Count_Enable = high

If (Test_acknowledge = high) then

Counter = Counter + one

If (Error = high) then

Counter = Counter + one

{'Fault in Processor #'Counter}

System HALT!

If (Count_Enable_Return = high) then

{'Test Complete'}

Load_Subgrids_line = high

Test_line = low

If (Test = no) then

```
        Load_Subgrids_line = high

    END TEST

                    {LOAD SUBGRIDS STATE}

3. If (Load_Subgrids_line = high) then

    Count_Enable = high

    While (Data_Needs_Loading = yes) do

        While (Error = yes) do

            {Wait for data from Bus Master}

            If (Master = message) then

                {Is message for load of subgrids?}

                If (Downloading_Subgrids = yes) then

                    {Master is writing in Controller's Local Memory}

                Else (Doing_Anything_Else = Error)

                    {'Improper Load. Input not accepted'}

            End ERROR Loop

            While (Wait = high) do

                {Do nothing until wait = low}

            Write_Enable = high

            While (Subgrids_to_Load) do

                {Subgrids written into an algorithm

                component processor's memory}.

            Write_Enable = low

            If (Count_Enable_Return = high) then

                Load_Subgrid_line = low

                Compute_Contour_line = high

            End While Loading Loop

    END LOAD SUBGRIDS
```

(COMPUTE CONTOURS STATE)

```
4. If (Compute_Contours_line = high) then
        First_Contour_Level = true
        While (Contour_Level = no) do
            {Do nothing until Contour_Level = yes}
            {In order for the contour level to be yes
            the bus master must send the controller the
            contour level and the controller must
            acknowledge its receipt.}
        {Contour_level = yes}
        Input_Count_Enable_line = high
        While (Input_Count_Enable_Return = low) do
            While (Wait_line = high) do
                {Do nothing until Wait_line = low}
            Write_Enable = high
            {Write Contour Level on to Input Bus}
            Write_Enable = low
            End ONE load cycle
            {Continue loading contour levels until the
            Input_Count_Enable_Return = high}
            If (First_Contour_Level = true) then
                Output_Count_Enable = high
                {After the initial start of the
                Output_Count_Enable, the
                Output_Count_Enable_Return acts as
                an interrupt to the systems controller
                and restarts the Output_Count_Enable
```

on each interrupt.}

End First Contour Level

End While Loop

END COMPUTE CONTOURS STATE

10. Appendix B

Algorithm Describing the Algorithm Component Processor

The following is a description of the algorithm component processor's behavior. It specifies this behavior as a high level abstract of the actual microcoding needed for the components implementation.

```
                                {RESET STATE}
1) If (Reset_line = high) then
        {Initialization of algorithm component processor}
        RAM_memory = zero
        Registers = zero

                                {TEST STATE}
2) If (Test_line = high) then
        {Hardware diagnostic test}
        If (No_errors) then
            While (Count_enable = low) do
                {Do nothing}
            {Count_enable = high}
            Test_acknowledge = high
            Count_enable_out = high
            End TEST
        Else {Error Condition}
            While (Count_enable = low) do
                {Do nothing}
            Error_line = high
            End TEST
```

{LOAD SUBGRIDS STATE}

3) If (Load_subgrids_line = high) then

Memory_pointer = start_RAM_address

While (Count_enable = low) do

{Do nothing}

{Count_enable = high}

Wait = high

If (Ready_to_load) then

Wait = low

While (Write_Enable = high) do

{Load subgrids into RAM}

{Write_enable = low}

Count_enable_out = high

End LOAD SUBGRIDS

{COMPUTE CONTOURS STATE}

4) If (Compute_contours_line = high) then

{Execute Contouring Algorithm (Appendix C)}

{Interrupts for Input/Output}

5) Input Interrupt:

Registers = zero

Wait = high

If (Ready_to_receive) then

Wait = low

If (Write_enable = high) then

{Transfer contour level}

Input_count_enable_out = high

```
        End Input Interrupt

6) Output Interrupt: {Output_count_enable = high}
        Stack = Push(Registers)
        If (Output) then
            #_bytes_to_transfer =
                End_address - Start_address
            ge.req = high
            While (ge.ack = high) do
                {Transfer Coordinates and Drawing Inst.}
            ge.req = low
            {End of Data transfer}
        Registers = Pop(Stack)
        Output_count_enable_out = high
        End Output Interrupt
```

11. Appendix C

The Contouring Algorithm

The Contouring Algorithm uses the subgrid definitions and contour levels to generate a three-dimensional contour surface display. A detailed description of the algorithm is included in [5].

{THE CONTOURING ALGORITHM}

For i = 1 to #_subgrids_in_memory {1514}

If (contour_level ^ = within_subgrid(i)) then

{Go on to next subgrid}

Else {contour level is within subgrid(i)}

{Compute 8 bit Adjacency Matrix index corresponding to subgrid (i).}

{For subgrid(i) generate the corresponding coordinates and drawing instructions from traversal lists.}

End Subgrid i Loop

END CONTOURING ALGORITHM

Dr. Henry Fuchs,
208 New West Hall (035A),
University of North Carolina,
Chapel Hill, NC 27514

Dr. Kent R. Wilson,
University of California, San Diego
B-014,
Dept. of Chemistry,
La Jolla, CA 92093

Dr. Guy L. Tribble, III
Apple Computer,
20525 Mariani Ave,
Cupertino, CA 95014

Dr. Victor Lesser,
University of Massachusetts, Amherst
Dept. of Computer and Information Science,
Amherst, MA 01003

Dr. Gunther Schrack,
Dept. of Electrical Engineering,
University of British Columbia,
Vancouver, B.C., Canada V6T 1W5

Dr. R. Daniel Bergeron,
Dept. of Computer Science,
University of New Hampshire,
Durham, NH 03824

Dr. Ed Wegman,
Division Head,
Mathematical Sciences Division,
Office of Naval Research,
800 N. Quincy Street,
Arlington, VA 22217-5000

Dr. Gregory B. Smith,
ATT Information Systems,
190 River Road,
Summit, NJ 07901

Dr. Lynn Conway,
Defense Advanced Research Projects Agency/IPTO,
1400 Wilson Blvd.,
Arlington, VA 22209

Dr. John Lowrance,
SRI International,
333 Ravenswood Ave.
Menlo Park, CA 94025

Dr. David Mizell,
Office of Naval Research,
1030 E. Green St.
Pasadena, CA 91106

Dr. Richard Lau,
Office of Naval Research,
Code 411,
800 N. Quincy St.
Arlington, VA 22217-5000

Dr. Y.S. Wu,
Naval Research Laboratory,
Code 7007,
Washington, D.C. 20375

Dr. Joel Trimble,
Office of Naval Research,
Code 251,
Arlington, VA 22217-5000

Robert A. Ellis,
Calma Company,
527 Lakeside Drive,
Sunnyvale, CA 94086

Dr. James H. Clark,
Silicon Graphics, Inc.
630 Clyde Court,
Mountain View, CA 94043

Shinji Tomita,
Dept. of Information Science,
Kyoto University,
Sakyo-ku, Kyoto. 606, Japan

Hiroshi Hagiwara,
Dept. of Information Science,
Kyoto University,
Sakyo-ku, Kyoto, 606, Japan

Dr. Alain Fournier,
Dept. of Computer Science,
University of Toronto,
Toronto, Ontario, Canada
M5S 1A4

Dr. Andries Van Dam,
Dept. of Computer Science,
Brown University,
Providence, RI 02912

Dr. Brian A. Barsky,
Berkeley Computer Graphics Laboratory,
Computer Sciences Division,
Dept. of Electrical Engineering and Computer Sciences,
University of California,
Berkeley, CA 94720

Dr. Ivan E. Sutherland,
Carnegie Mellon University,
Pittsburg, PA 15213

Dr. Turner Whitted,
208 New West Hall (035A),
University of North Carolina,
Chapel Hill, NC 27514

Dr. Robert B. Grafton,
Office of Naval Research,
Code 433,
Arlington, Virginia 22217-5000

Professor Eihachiro Nakamae,
Electric Machinery Laboratory,
Hiroshima University,
Higashihiroshima 724, Japan

Carl Machover,
Machover Associates,
199 Main Street,
White Plains, New York 10601

Dr. Buddy Dean,
Naval Postgraduate School,
Code 52, Dept. of Computer Science,
Monterey, California 93943

Earl Billingsley,
43 Fort Hill Terrace,
Northhampton, MA 01060

Dr. Jan Cuny,
University of Massachusetts, Amherst
Dept. of Computer and Information Science,
Amherst, MA 01003

Robert Lum,
Silicon Graphics, Inc.
630 Clyde Court,
Mountain View, CA 94043

Jeff Hausch,
Silicon Graphics, Inc.
630 Clyde Court,
Mountain View, CA 94043

Lt. Robert A. Walker,
Naval Sea Systems Command (SEA 61YM),
Department of the Navy,
Washington, DC 20362-5101

Marvin Katich,
Silicon Graphics, Inc.
630 Clyde Court,
Mountain View, CA 94043

Dr. Barry L. Kalman,
Washington University,
Department of Computer Science.
Box 1045,
St. Louis, Missouri 63130

Dr. Wm. Randolph Franklin,
Electrical, Computer, and Systems Engineering Department,
Rensselaer Polytechnic Institute,
Troy, New York 12180-3590

Dr. Gershon Kedem,
Microelectronics Center of North Carolina,
PO Box 12889,
3021 Cornwallis Road,
Research Triangle Park.
North Carolina 27709

Dr. Branko J. Gerovac,
Digital Equipment Corporation.
150 Locke Drive LMO4/H4. Box 1015
Marlboro, Massachusetts 01752-9115

Robert A. Schumacker,
Evans and Sutherland,
PO Box 8700,
580 Arapeen Drive,
Salt Lake City, Utah 84108

R. A. Dammkoehler,
Washington University,
Department of Computer Science,
Box 1045,
St. Louis, Missouri 63130

	No. Of Copies
Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Library, Code 0142 Naval Postgraduate School Monterey, CA 93943	2
Center for Naval Analyses 2000 N. Beauregard Street Alexandria, VA 22311	1